National Research Council Canada
Conseil national de recherches Canada

Division of Electrical Engineering
Division de génie électrique

ERB-1009

A New Algorithm for Point and Plane Pattern Matching

S. Umeyama and T. Kasvand

November 1987

Canada

NRCC No. 28455

88-04447/

Copyright 1987 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright 1987 par
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures ou tableaux du présent rapport, à condition d'en identifier clairement la source.

Additional copies are available free of charge from:

Editorial Office, Room 301
Division of Electrical Engineering
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R8

Des exemplaires supplémentaires gratuits peuvent être obtenus à l'adresse suivante:

Bureau des publications, Pièce 301
Division de génie électrique
Conseil national de recherches du Canada
Ottawa (Ontario) Canada
K1A 0R8

A NEW ALGORITHM FOR POINT AND PLANE PATTERN MATCHING

S. Umeyama and T. Kasvand

ERB-1009

NRCC No. 28455

November 1987

TABLE OF CONTENTS

Page

ABSTRACT v
1. INTRODUCTION 1
2. POINT PATTERN MATCHING 3
2.1 Definition of a Problem 3
2.2 Point Pattern Matching Algorithm 4
2.2.1 Pruning by the Eligible Interval of Scaling Factor of a Matching 5
2.2.2 Pruning by the Least-mean-squared Error of a Matching 7
2.2.3 Tree Search Algorithm 11
2.3 Control of the Search by Geometric Constraint 13
2.4 Performance Evaluation of the Point Pattern Matching Algorithm 14
2.4.1 Matching of Randomly Generated Point patterns . . . 15
2.4.2 Point Pattern Matching in 3-D Space 17
2.4.3 Shape Matching 18
3. PLANE PATTERN MATCHING 21
3.1 Definition of a Problem 21
3.2 Plane Pattern Matching Algorithm 23
4. CONCLUSION 25
ACKNOWLEDGEMENTS 25
REFERENCES 25
APPENDIX A. Proof of Theorem 1 27
APPENDIX B. Proof of Theorem 2 33
APPENDIX C. Point Pattern Matching Program for 2-D 35
APPENDIX D. Point Pattern Matching Program for 3-D 47
APPENDIX E. Shape Matching Program 59

List of Figures

1 Matching of point patterns (n = 3) 4
2 A δ-bounded matching 4
3 Search tree for matching between point pattern X and Y (|X| = 2,|Y| = 3) 6
4 Eligible interval of scaling factor of a matching $f_2 = ((x_i, y_i), (x_j, y_j))$ 6
5 Tree search algorithm for point pattern matching 12
6 Search tree pruned by the size restriction (|X| = 4, N = 3) 13

TABLE OF CONTENTS (Concluded)

List of Figures (Concluded)

		Page
7	Randomly generated point patterns in 2-D space	15
8	Matching results	16
9	Randomly generated point patterns in 3-D space (projection to X-Y plane)	17
10	An object used as a model	17
11	Matching result	18
12	Polygonal approximation of a contour of overlapping objects	19
13	Polygonal approximations of model contours	19
14	Inconsistent matching with respect to the order relation of corners	20
15	Inner angles at corners and a valid correspondence	20
16	Matching results for shape matching	21
17	Representation of a plane	22

ABSTRACT

This report presents an algorithm for finding a matching between two point or plane patterns given in m-dimensional Euclidean space. This matching algorithm is formulated as a tree search procedure, and the search tree is pruned by a specified tolerance measure for matching. The proposed method is based on the locations and is invariant to rotation, translation, and scaling. Moreover, it is easy to use additional information (such as semantic labels or order relations for points) within the same framework in order to guide the search. Global geometric constraints known in advance, such as rotation or scaling parameters, can also be exploited easily. The performance of the algorithm is demonstrated in the case of point pattern matching by matching of randomly generated point patterns and shape matching for real scenes.

RÉSUMÉ

Le présent rapport présente un algorithme de couplage entre deux combinaisons de points ou de plans données dans un espace euclidien de dimension m. Cet algorithme est présenté sous forme d'une méthode de recherche arborescente, et l'arbre de recherche est émondé par une mesure de tolérence spécifiée pour le couplage. La méthode proposée est basée sur les positions et est indépendante de la rotation, de la translation et de la mise à l'échelle. De plus, il est facile d'utiliser des informations additionnelles (par exemple des marques sémantiques ou des relations d'ordre pour les points) à l'intérieur de la même structure afin de guider la recherche. Les constraintes géométriques connues à l'avance, par exemple les paramètres de rotation ou de mise à l'échelle, peuvent aussi être exploitées facilement. On démontre l'application de l'algorithme dans le cas d'une combinaison de points en effectuant le couplage de combinaisons de points générées de façon aléatoire et le couplage de formes pour des scènes réelles.

A NEW ALGORITHM FOR POINT AND PLANE PATTERN MATCHING

S. Umeyama and T. Kasvand

1. INTRODUCTION

Finding the best matching between two images or between an object and a known model is one of the most important problems in computer vision and image analysis. For example, in a widely-accepted paradigm for computer vision, model matching is the final stage of a four-step process: (1) restoration (cleaning up the image), (2) segmentation (partitioning into significant regions), (3) description (extracting local features), and (4) model matching (1, 2). Another example is in motion analysis, where motion parameters are estimated by matching images of successive frames.

A commonly used approach to matching is to extract a set of local features (e.g., edges or corners) from each image and find a good correspondence between them. Features usually have a variety of properties such as locations, semantic labels, or order relations (e.g., along a boundary) (3). In this report, however, we focus on the utilization of location, which is regarded as the most common and important property of features. We consider that features are represented as unlabeled geometric entities (such as a point or a plane) in Euclidean space. The problem consists then of matching two sets of such entities, X and Y, in m-dimensional space, which can be also viewed as a problem of finding a "rigid" transformation (rotation, translation, and scaling) which will superpose X on Y as closely as possible. We adopt a point and a plane as a geometric entity in this report. A set of points (planes) in Euclidean space is called a point (plane) pattern (4, 5), and a matching of two sets of points (planes) is a point (plane) pattern matching problem.

In the literature, many matching strategies are used with different kinds of allowed transformation, although almost all of them deal with a point pattern matching problem. Ranade and Rosenfeld (4) applied a relaxation method to this problem. A figure of merit of each correspondence between a point in X and a point in Y is rated by its affect on other points, and this value is updated iteratively on the basis of local evidence (relaxation process). Their method allows translation between point patterns, although it is not applicable to rotated patterns. Baird (2) employed a combinatorial (a tree search) approach to find a matching between two point patterns with the same number of points. The Soviet ellipsoid algorithm (6) was used to prune a search tree. His method can cope

with general rigid transformations in two-dimensional space since this can be represented in linear form with respect to transformation parameters. However, it seems difficult to extend his method to the three-dimensional case since the transformations are no longer linear. His method can be extended to the cases with spurious and missing points. Wong (7) gives another tree search method. He employed a graph representation of a constellation (point pattern) and applied the branch-and-bound heuristics to guide the tree search for the best matching of graphs. The "(n, k)-similarity" between two constellations is defined in order to prune the search tree further. His method is applicable to constellation matching problems in any metric space and seems very efficient. However, it is not invariant to scaling, and it cannot distinguish the reflection (mirror image) of a pattern from the original image since they are identical in graph representation. Other approaches and related results of point pattern matching can be found in refs. 5, 8, and 9.

In this report, we present a new algorithm for finding a matching between two point or plane patterns given in m-dimensional Euclidean space. A plane pattern matching is important especially when handling problems in three-dimensional space since surface information is more stable and reliable than feature point information in such cases. The algorithm is formulated as a tree search procedure, and the search tree is pruned by a specified tolerance measure for matching. The tolerance measure is defined as the maximum distance between the transformed entities and their corresponding entities. The proposed method is based on the locations and is invariant to rotation, translation, and scaling. Moreover, it is easy to use additional information (such as semantic labels or order relations of points) within the same framework to guide the search. Global geometric constraints such as rotation or scaling parameters known in advance can also be exploited easily.

We describe an algorithm for a point pattern matching problem in detail and give a brief discussion of a plane pattern matching problem. Section 2.1 defines a point pattern matching problem, and Section 2.2 gives a tree search algorithm for that problem and its search tree pruning methods. Search control methods by geometric constraints are described in Section 2.3, and the performance evaluation of the proposed algorithm is given in Section 2.4. Sections 3.1 and 3.2 gives a definition of a plane pattern matching problem and its tree search algorithm, respectively. Finally, Section 4 gives the conclusion. This report also includes 2-D and 3-D point pattern matching programs and a 2-D shape matching program written in Pascal in Appendixes C, D, and E.

2. POINT PATTERN MATCHING

2.1 Definition of a Problem

A point pattern X is a set of points in m-dimensional Euclidean space, and we write $|X|$ for the number of points in X. A matching of size n ($n \leq |X|$ and $n \leq |Y|$) between two point patterns, X and Y, is a one-to-one mapping f from a subset X_f of X of size n onto a subset Y_f of Y of the same size (Fig. 1). f(x) is a point in Y corresponding to x. A sub-matching g of a matching f is a matching from $X_g \subseteq X_f$ to $Y_g \subseteq Y_f$ and satisfying $g(x) = f(x)$ if $x \in X_g$.

A registration T is a rigid transformation from m-dimensional space onto itself, consisting of a composition of translation, rotation, and non-negative scaling (2). A point x transformed by a registration T is represented as follows:

$$T(x) = cRx + t \qquad [1]$$

where R, t, and c are a rotation matrix (a proper orthogonal matrix, i.e., $R'R = I$, $\det(R) = 1$, a translation vector, and a scaling factor, respectively.

We say that a matching f of size n is δ-bounded if the following holds,

$$d = \max_{x \in X_f} ||f(x) - T_f(x)|| \leq \delta \qquad [2]$$

where $||.||$ is the Euclidean norm ($||x|| = (\sum_{i=1}^{m} x_i^2)^{1/2}$), δ is a prespecified tolerance, and T_f is an optimum registration from X_f to Y_f in the sense of the least-mean-squared error, i.e., T_f is a registration T which minimizes the following error term e:

$$e = (1/n) \sum_{x \in X_f} ||f(x) - T(x)||^2 \qquad [3]$$

We call d defined in eq. [2] the maximum error distance of f, and write ε for the minimum value of e, i.e., the least-mean-squared error. Note that we choose to measure errors in the scale for Y. The definition of δ-boundedness means that, when a point pattern X is superposed on a point pattern Y by the optimum registration T_f, the maximum distance between the corresponding points in X_f and Y_f is less than or equal to δ (see Fig. 2).

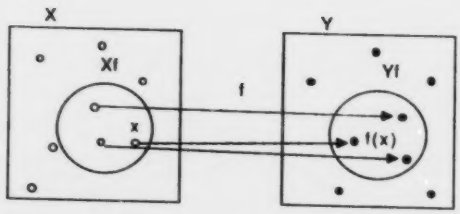


Figure 1. Matching of point patterns (n = 3).

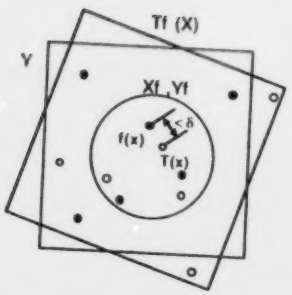


Figure 2. A δ-bounded matching.

Now, a point pattern matching problem between two point patterns X and Y is defined as a problem of finding all δ-bounded matchings of the maximum size N. N is the maximum number of points that can be matched under the condition of δ-boundedness.

2.2 Point Pattern Matching Algorithm

The selection of a desired matching for a point pattern matching problem is formulated as a tree search. The tree search algorithm is described in this section.

The algorithm conducts a search through a tree which represents every possible matching as a path from the root to a leaf. Given a point pattern X, a comparison order on the points in X is chosen, say: $x_1, x_2, \ldots$. This order will remain fixed during matching, and points in X are compared with points in Y in this order. Points in Y are presented, of course, in an arbitrary order: $y_1, y_2, \ldots$. Figure 3 shows an example of a search tree when $|X| = 2$ and $|Y| = 3$. Each node of the tree represents a correspondence between a point in X (indicated by the depth of the node) and a point in Y (indicated by the label of the node). Every possible matching is represented by a path from the root to a certain leaf, and a path from the root to any node on this path defines a partial matching of this matching. A partial matching is considered to be a sub-matching of the original matching. For example, a path P in Fig. 3 represents a matching $((x_1, y_2), (x_2, y_1))$, and a partial path P' of P defines a sub-matching $((x_1, y_2))$ of this matching. Note that many nodes have "*" as their labels. This indicates that the point in X corresponds to no point in Y, and thus provides a means of excluding points from a matching. A path Q, for example, represents a matching $((x_1,*), (x_2, y_3) = (x_2, y_3))$. Exploration of a tree is done in depth-first order, so that, for any matching $f = ((x_1, f(x_1)), (x_2, f(x_2)), \ldots)$ $(f(x_i) \in Y$ or $f(x_i) =$ "*"), all partial matchings $((x_1, f(x_1)), \ldots, (x_k, f(x_k)))$ $(k < n)$ of f are examined earlier than f.

In order to find a δ-bounded matching of the maximum size, the δ-boundedness of all possible matchings must be tested. This is, however, practically impossible since the number of possible matchings is enormous and it increases very rapidly as the size of the pattern increases. Hence, efficient pruning methods are essential for a practical matching algorithm.

Two kinds of search tree pruning methods are employed in our algorithm. Both methods are based on conditions that all sub-matching of a δ-bounded matching should satisfy. Thus, if a partial matching does not satisfy one of these conditions, all matchings including this partial matching cannot be δ-bounded and need not be examined further. The sub-tree with this partial matching as its root can be pruned in this case and all matchings represented by this sub-tree will be discarded.

As shown later, these conditions can be computed recursively as the depth-first search proceeds further. This recursive computability is very important since computation time of the algorithm is determined by two factors: the size of the tree and the cost of testing conditions of δ-boundedness. If we use the depth-first search and the pruning method given in the following sections, the size of the tree is determined only by the geometry of the point patterns and the comparison order of points in X, and these are fixed when two point patterns are given. Thus, reducing the cost of testing is crucial for the efficiency of the algorithm.

2.2.1 Pruning by the Eligible Interval of Scaling Factor of a Matching

Let g and f be a δ-bounded matching and its sub-matching between two point patterns, X and Y, respectively, and let T_g be the optimum registration of g. Now we consider a sub-matching $f_2 = ((x_i, y_i), (x_j, y_j))$ of f of size two. Obviously, from Fig. 4, if the scaling factor of a registration from X to Y is too large or too small, it is impossible to transform x_i and x_j by this registration into spheres with centers of y_i and y_j and radius of δ, even if the rotation and translation of the registration could be chosen without any restriction. More precisely, any registration cannot put x_i and x_j into these spheres at the same time if its scaling factor is on the outside of the interval $E(f_2)$ defined in ref. 4.

[4] $$E(f_2) = E(((x_i, y_i), (x_j, y_j))) = [c_min, c_max]$$

where

$$c_min = \max(0, (||y_j - y_i|| - 2\delta)/||x_j - x_i||$$

$$c_max = (||y_j - y_i|| + 2\delta)/||x_j - x_i||$$

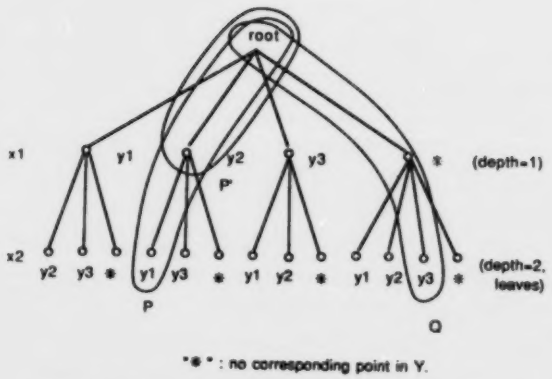


Figure 3. Search tree for matching between point pattern X and Y ($|X| = 2$, $|Y| = 3$).

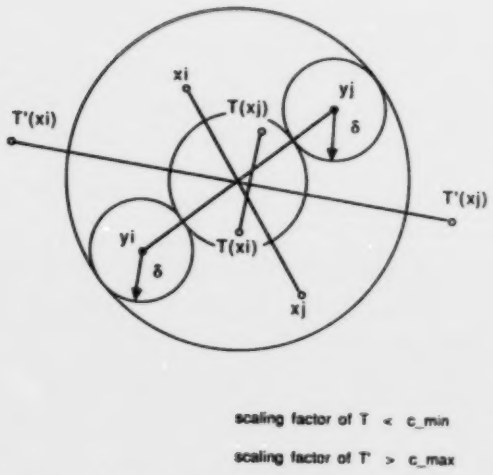


Figure 4. Eligible interval of scaling factor of a matching $f_2 = ((x_i, y_i), (x_j, y_j))$.

We call this interval the eligible interval of scaling factor of a matching f_2. Thus, since g is a δ-bounded matching, the scaling factor c of T_g must be within this interval.

This kind of constraint should be satisfied for all sub-matchings of f of size two so that f will be a sub-matching of a δ-bounded matching g. Hence, c should be within the interval E(f) derived as the intersection of all such intervals.

$$[5] \qquad c \in E(f) = \bigcap_{f \supseteq f_2 \text{ and size } (f_2)=2} E(f_2)$$

where "$\bigcap$" means the intersection of intervals.

This means, in other words, that if the eligible interval E(f) is an empty set, f can never be a sub-matching of any δ-bounded matching. This property can be used to prune the search tree. If the eligible interval E(f) of a partial matching f is found to be an empty set, all matchings including this partial matching as a sub-matching can be discarded without further testing.

The eligible interval of a matching can be computed recursively as:

$$[6a] \qquad E(f_1) = [0, \text{infinity})$$

$$[6b] \qquad E(f_{n+1}) = E(f_n) \bigcap \bigcap_{i=1,n} E((x_i, y_i), (x_{n+1}, y_{n+1}))$$

where

$$f_n = ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$$

$$f_{n+1} = f_n \cup (x_{n+1}, y_{n+1})$$

and $E((x_i, y_i), (x_{n+1}, y_{n+1}))$ is defined in eq. [4].

2.2.2 Pruning by the Least-mean-squared Error of a Matching

The search tree can be pruned also by the value of the least-mean-squared error of a partial matching. Let X and Y be point patterns and f be a matching of size n between X and Y. If the least mean squared error ε of f is greater than δ^2, any matching g which includes f as a sub-matching cannot be δ-bounded. Thus, the sub-tree with this partial matching as its root can be pruned similarly to the previous section.

Now we show this property. Let T_f and T_g be the optimum registrations between X_f and Y_f and between X_g and Y_g, respectively ($X_f \subseteq X_g$, $Y_f \subseteq Y_g$). Assume that g is δ-bounded. Then

$$[7] \qquad \max_{x \in X_g} ||g(x) - T_g(x)|| < \delta$$

Since $X_f \subseteq X_g$ and $f(x) = g(x)$ for $x \in X_f$, we have

$$[8] \qquad \max_{x \in X_f} ||f(x) - T_g(x)|| = \max_{x \in X_g} ||g(x) - T_g(x)||$$

$$\leq \max_{x \in X_g} ||g(x) - T_g(x)||$$

$$< \delta$$

Thus

$$[9] \qquad (1/n) \sum_{x \in X_f} ||f(x) - T_g(x)||^2 < \delta^2$$

On the other hand, since we assumed that the least-mean-squared error ε of f is greater than δ^2, the following holds,

$$[10] \qquad \varepsilon = (1/n) \sum_{x \in X_f} ||f(x) - T_f(x)||^2 > \delta^2$$

T_g may not be the optimum registration between X_f and Y_f; thus we have

$$[11] \qquad (1/n) \sum_{x \in X_f} ||f(x) - T_g(x)||^2 > \varepsilon$$

Hence, from eqs. [10] and [11]

$$[12] \qquad (1/n) \sum_{x \in X_f} ||f(x) - T_g(x)||^2 > \delta^2$$

This contradicts eq. [9]. Thus g is not δ-bounded.

The least-mean-squared error ε of a matching is given by the following theorem.

Theorem 1: Let f be a matching of size n between point patterns X and Y in m-dimensional space and $X_f = \{x_1, x_2, \ldots, x_n\}$ and $Y_f = \{f(x_1), f(x_2), \ldots, f(x_n)\} = \{y_1, y_2, \ldots, y_n\}$. Then the least-mean-squared error ε is given as follows:

$$[13] \qquad \varepsilon = s_y^2 - [\mathrm{tr}(DS)]^2/s_x^2$$

where

$$[14a] \qquad m_x = (1/n) \sum_{i=1}^{n} x_i, \qquad m_y = (1/n) \sum_{i=1}^{n} y_i$$

$$[14b] \qquad s_x^2 = (1/n) \sum_{i=1}^{n} ||x_i - m_x||^2$$

$$= (1/n) \sum_{i=1}^{n} ||x_i||^2 - ||m_x||^2$$

$$[14c] \qquad s_y^2 = (1/n) \sum_{i=1}^{n} ||y_i - m_y||^2$$

$$= (1/n) \sum_{i=1}^{n} ||y_i||^2 - ||m_y||^2$$

$$[14d] \quad S_{xy} = (1/n) \sum_{i=1}^{n} (y_i - m_y)(x_i - m_x)'$$

$$= (1/n) \sum_{i=1}^{n} y_i x_i' - m_y m_x'$$

$$= UDV' \quad : \text{singular value decomposition of } S_{xy}$$

$$D = \text{diag}(d_i),\ d_1 \geq d_2 \geq \ldots \geq d_m \geq 0$$

$$[14e] \quad S = \begin{cases} I & \text{if } \det(S_{xy}) \geq 0 \\ \text{diag}(1, 1, \ldots, 1, -1) & \text{if } \det(S_{xy}) < 0 \end{cases}$$

where "'" means the transpose of a vector or a matrix. S_{xy} is a covariance matrix of X_f and Y_f, m_x and m_y are mean vectors of X_f and Y_f, and s_x^2 and s_y^2 are variances around the mean vectors of X_f and Y_f, respectively. When $\text{rank}(S_{xy}) = m$, the optimum registrations T_f (R, t, c) are determined uniquely as follows:

$$[15a] \quad R = USV'$$

$$[15b] \quad t = m_y - cRm_x$$

$$[15c] \quad c = \text{tr}(DS)/s_x^2$$

The proof of the theorem is given in Appendix A.

Next we show that the characteristics defined in eq. [14] can be computed recursively. Let f_n be a matching of size n between point patterns X and Y, and f_{n+1} be an augmented matching of size n + 1 created by adding a new correspondence (x_{n+1}, y_{n+1}) to f_n. Then the characteristics $(m_{x,n+1}, m_{y,n+1}, s^2_{x,n+1}, s^2_{y,n+1}, S_{xy,n+1})$ of f_{n+1} in eq. [14] can be computed recursively from the characteristics $(m_{x,n}, m_{y,n}, s^2_{x,n}, s^2_{y,n}, S_{xy,n})$ of f_n.

Obviously we have

$$[16a] \quad m_{x,n+1} = (n/(n+1))(m_{x,n} + x_{n+1}/n)$$

$$[16b] \quad m_{y,n+1} = (n/(n+1))(m_{y,n} + y_{n+1}/n)$$

Next,

$$[17] \quad (n+1)s^2_{x,n+1} = n(s^2_{x,n} + ||m_{x,n}||^2) + ||x_{n+1}||^2 - (n+1)||m_{x,n+1}||^2$$

$$= ns^2_{x,n} + n||m_{x,n}||^2 + ||x_{n+1}||^2 - (n^2||m_{x,n}||^2 + 2nm'_{x,n}x_{n+1} + ||x_{n+1}||^2)/(n+1)$$

$$= ns^2_{x,n} + n(||m_{x,n}||^2 - 2m'_{x,n}x_{n+1} + ||x_{n+1}||^2)/(n+1)$$

$$= ns^2_{x,k} + n||m_{x,n} - x_{n+1}||^2/(n+1)$$

Thus

$$[18a] \quad s^2_{x,n+1} = [n/(n+1)][s^2_{x,n} + ||m_{x,n} - x_{n+1}||^2/(n+1)]$$

$$[18b] \quad s^2_{y,n+1} = [n/(n+1)][s^2_{y,n} + ||m_{y,n} - y_{n+1}||^2/(n+1)]$$

Finally,

$$[19] \quad (n+1)S_{xy,n+1} = n(S_{xy,n} + m_{y,n}m'_{x,n}) + y_{n+1}x'_{n+1} - (n+1)m_{y,n+1}m'_{x,n+1}$$

$$= nS_{xy,n} + nm_{y,n}m'_{x,n} + y_{n+1}x'_{n+1} - (nm_{y,n} + y_{n+1})(nm_{x,n} + x_{n+1})'/(n+1)$$

$$= nS_{xy,n} + n(m_{y,n}m'_{x,n} - m_{y,n}x'_{n+1} - y_{n+1}m'_{x,n} + y_{n+1}x'_{n+1})/(n+1)$$

$$= nS_{xy,n} + n(m_{y,n} - y_{n+1})(m_{x,n} - x_{n+1})'/(n+1)$$

Thus

$$[20] \quad S_{xy,n+1} = [n/(n+1)][S_{xy,n} + (m_{y,n} - y_{n+1})(m_{x,n} - x_{n+1})'/(n+1)]$$

Hence, if these characteristics are stored in every expanded node, the least-mean-squared error ε and the optimum registration T_f can be computed

very efficiently. This will reduce the cost of testing δ-boundedness and contributes the efficiency of the algorithm.

Finally, we give some remarks about the computation of $(tr(DS))^2$ in eq. [13]. In general, we need to compute the singular value decomposition of S_{xy} in order to obtain the value $(tr(DS))^2$ in eq. [11]. This computation is not difficult since the size of the matrix is usually very small (2 or 3) in real applications. Moreover, when point patterns are planar, i.e., they are point patterns in two-dimensional space, we can reduce the amount of computation further. In this case, from eq. [14],

$$[21] \qquad (tr(DS))^2 = \begin{cases} (d_1 + d_2)^2 & \text{if } \det(S_{xy}) > 0 \\ (d_1 - d_2)^2 & \text{if } \det(S_{xy}) < 0 \end{cases}$$

Obviously we have,

$$[22] \qquad tr(S_{xy}S'_{xy}) = d_1^2 + d_2^2$$

$$[23] \qquad \det(S_{xy}) = \begin{cases} d_1 d_2 & \text{if } \det(S_{xy}) > 0 \\ -d_1 d_2 & \text{if } \det(S_{xy}) < 0 \end{cases}$$

Thus

$$[24] \qquad (tr(DS))^2 = tr(S_{xy}S'_{xy}) + 2\det(S_{xy})$$
$$= (s_{11} + s_{22})^2 + (s_{12} - s_{21})^2$$

where

$$[25] \qquad S_{xy} = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix}$$

2.2.3 Tree Search Algorithm

The tree search algorithm for a point pattern matching problem is as follows (see Fig. 5): Initially the maximum size N of δ-bounded matchings is assumed to be 2 (since any two points in X can be always matched with any two points in Y) and a MM list (a list of maximum matchings) is initialized to a null list. The MM list is used to store copies of currently maximum matchings (i.e., matchings of size N). If N = 2, all matchings of this size are δ-bounded; thus they are useless and we do not store these

matchings in the MM list. N is updated when a δ-bounded matching of the size greater than N was found. A search for a δ-bounded matching of size N and N + 1 is conducted in the depth-first manner. The eligible interval of a (partial) matching denoted by each node of the tree is computed, and if that interval becomes an empty set, this matching and its descendants are discarded. If the eligible interval was not empty, then the least-mean-squared error ε of the matching is computed and compared with the value δ^2. If ε is bigger than δ^2, this matching and its descendants are also discarded, since they cannot be δ-bounded.

The maximum size of a δ-bounded matching is at least N; thus the search tree is pruned further by using this restriction. For example, if |X| = 4 and N = 3, the sub-tree denoted by "A" in Fig. 6 is pruned since

```
Input
  X, Y      : point patterns (sets of points).

Output
  MM_list : A list of δ-bounded matchings of the maximum size between X and Y.

Algorithm

N := 2;  ON_list := f0;
   { f0 is a null matching, size(f0) = 0.}
   { ON_list is a open node list, i.e., a list of matchings waiting for expansion.}

while ON_list ≠ nil do
  begin
    f := get_one_matching(ON_list);
    f' := f ∪ (xn+1,*);  insert(f',ON_list);
          { xn+1 has no corresponding point in Y }
    for all points y in Y that do not appear in f do
      begin
        f' := f ∪ (xn+1,y);
        compute the eligible interval E(f');
        if E(f') = empty then discard(f') { pruning of the search tree }
         else begin
           compute the least mean squared error ε(f');
           if ε(f') > δ² then discard(f') { pruning of the search tree }
            else begin
                insert(f',ON_list);
                case size(f') of
                   N+1  : begin
                             compute the maximum error distance d of f';
                             if d < δ then
                                begin discard(MM_list); copy(f',f");
                                      insert(f",MM_list); N := N+1 end;
                          end;
                     N    : begin copy(f',f"); insert(f",MM-list) end;
                end;
            end;
         end;
      end;
  end;
output(MM_list);
```

Figure 5. Tree search algorithm for point pattern matching.

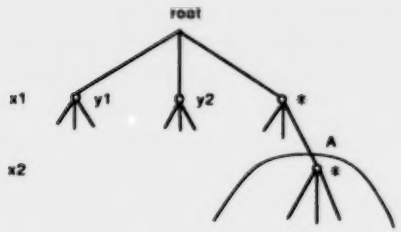


Figure 6. Search tree pruned by the size restriction ($|X| = 4$, $N = 3$).

the sizes of all matchings included in this sub-tree are obviously less than 3.

While searching a tree, if a matching f of size N + 1 has passed the above-mentioned tests, optimum registration T_f of this matching and the maximum error distance d in eq. [2] are actually computed in order to check the δ-boundedness of this matching. If this succeeds, the obtained matching is a δ-bounded matching greater in size than the matchings currently stored in the MM list. Hence, the current contents of the MM list are discarded and the obtained matching is copied and stored as a first element of the MM list. N is incremented.

If the size of a matching which passed these tests is N, this matching is supposed to be a δ-bounded matching without checking the actual δ-boundedness, i.e., without computing the maximum error distance d, and added to the MM list. We do not check the δ-boundedness of this matching since this matching may be discarded when a greater matching is found, and if so, the computation of this checking becomes useless. If, finally, no greater matching is found and the search tree is exhausted, all the matchings stored in the MM list are actually checked about δ-boundedness, and matchings which pass this check are presented as solutions of the problem. At least one matching passes this check since the first element of MM list has already been checked when it was stored there.

2.3 Control of the Search by Geometric Constraints

The proposed point pattern matching method places no constraint on the registration between two point patterns. In several applications, however, some constraints are naturally imposed on parameters of a registration. For example, two stereoscopic views of an object have no vertical translation with respect to each other, and in motion analysis, a registration between two successive frames may not change very much. Another example is model-based 3-D object recognition. Sometimes a recognized object has its own stable positions; thus these positions can be used to constrain possible rotations between a model and an object.

Our method can easily exploit such global geometric constraints to guide or further prune the generation of partial matchings. Assume that we know the estimated values of some constraints on the parameters of a registration between two point patterns in advance. In this case, registration parameters of each partial matching are computed according to the eq. [15], and the computed parameters are compared with the given constraints. If they do not satisfy the given constraints, this matching and its descendants are discarded. Thus only matchings satisfying the constraints are generated.

When a constraint about a scaling factor is given, we need a special treatment of this constraint, since we already have a constraint on a scaling factor: the eligible interval. For example, if a scaling factor of a registration between two point patterns must be within one of these intervals, $[c_{11}, c_{12}]$, $[c_{21}, c_{22}]$, ..., and $[c_{k1}, c_{k2}]$, then the initial eligible interval E(f) defined in [6a] should be restricted as follows:

$$[26] \qquad E(f) = [c_{11}, c_{12}] \cup [c_{21}, c_{22}] \cup \cdots \cup [c_{k1}, c_{k2}]$$

where "$\cup$" means the union of intervals. A scaling factor of each partial matching computed according to eq. [15c] is compared with an eligible interval of that matching. If it is not within the interval, this matching is judged to be unsuitable for the solution, and thus this matching and its descendants are discarded.

Note that the obtained matching may no longer be a maximum matching if this pruning of the tree is done, since a maximum matching might have a sub-matching which does not satisfy the given constraints even if the maximum matching itself satisfies the constraints. However, this method usually gives a good matching efficiently, as shown in examples in Section 2.4.

Our depth-first pruned search method can also provide many possibilities to use non-geometric properties of point patterns, such as semantic labels and order relations of points (e.g., along a boundary), to prune the search tree. This is also illustrated by an example of shape matching given in Section 2.4.

2.4 Performance Evaluation of the Point Pattern Matching Algorithm

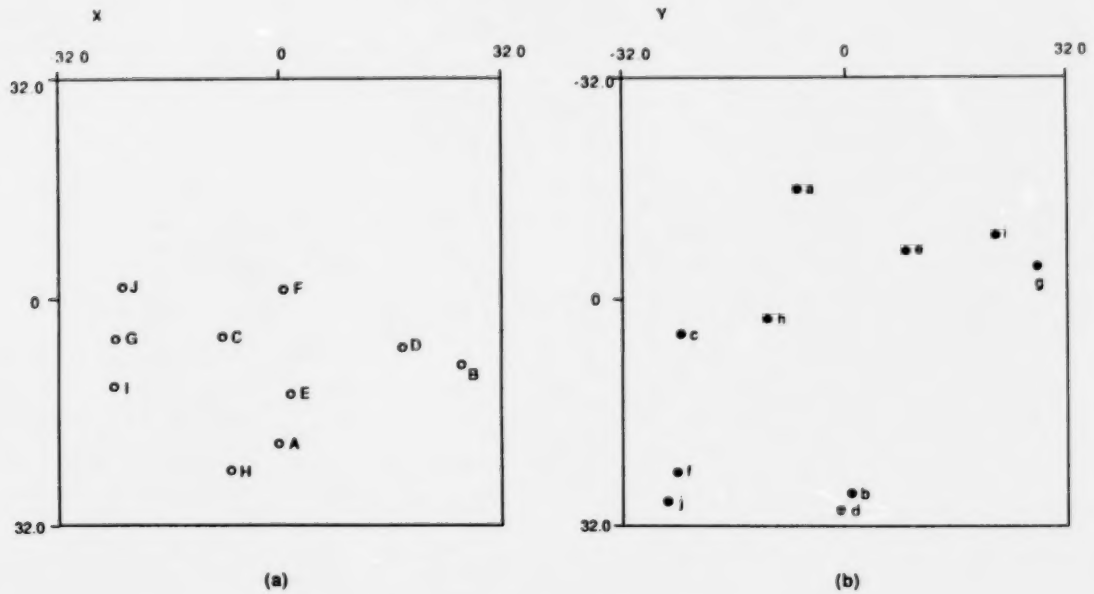
We present in this section three examples of the point pattern matching problem in order to illustrate the application and the performance evaluation of the proposed point pattern matching algorithm. The first example is matchings of randomly generated point patterns, and the second

example is the point pattern matching in 3-D space. The final example is the shape matching of machine parts.

2.4.1 Matching of Randomly Generated Point Patterns

Figure 7 a and b show randomly generated point patterns X and Y in two-dimensional space. Each point in X and Y was produced by a pseudo random number generator which assigned a real number uniformly distributed in the range of -32.0 to 32.0 to each coordinate of the point. Each pattern includes 10 points. These point patterns were matched by using the proposed point pattern matching algorithm. The matching results are shown in Fig. 8, where only matchings which have the least d (the maximum error distance) among the obtained matchings are shown, and the transformed X is superposed on Y. Below each result, parameters of an obtained registration and a list of matched point pairs are given. The maximum error distance d of the matching is also given below.

Figure 8 a and b show the matching results when $\delta = 5.0$ and $\delta = 3.0$. No constraint is imposed on parameters of a registration in this case. The sizes of the obtained matchings were 6 and 5, and the number of obtained matchings was 12 and 14, respectively. On the other hand, Fig. 8 c and d show the matching results when some constraint is imposed on a parameter. In Fig. 8c, a scaling factor c of a registration was given in advance,



(a) (b)

Figure 7. Randomly generated point patterns in 2-D space.

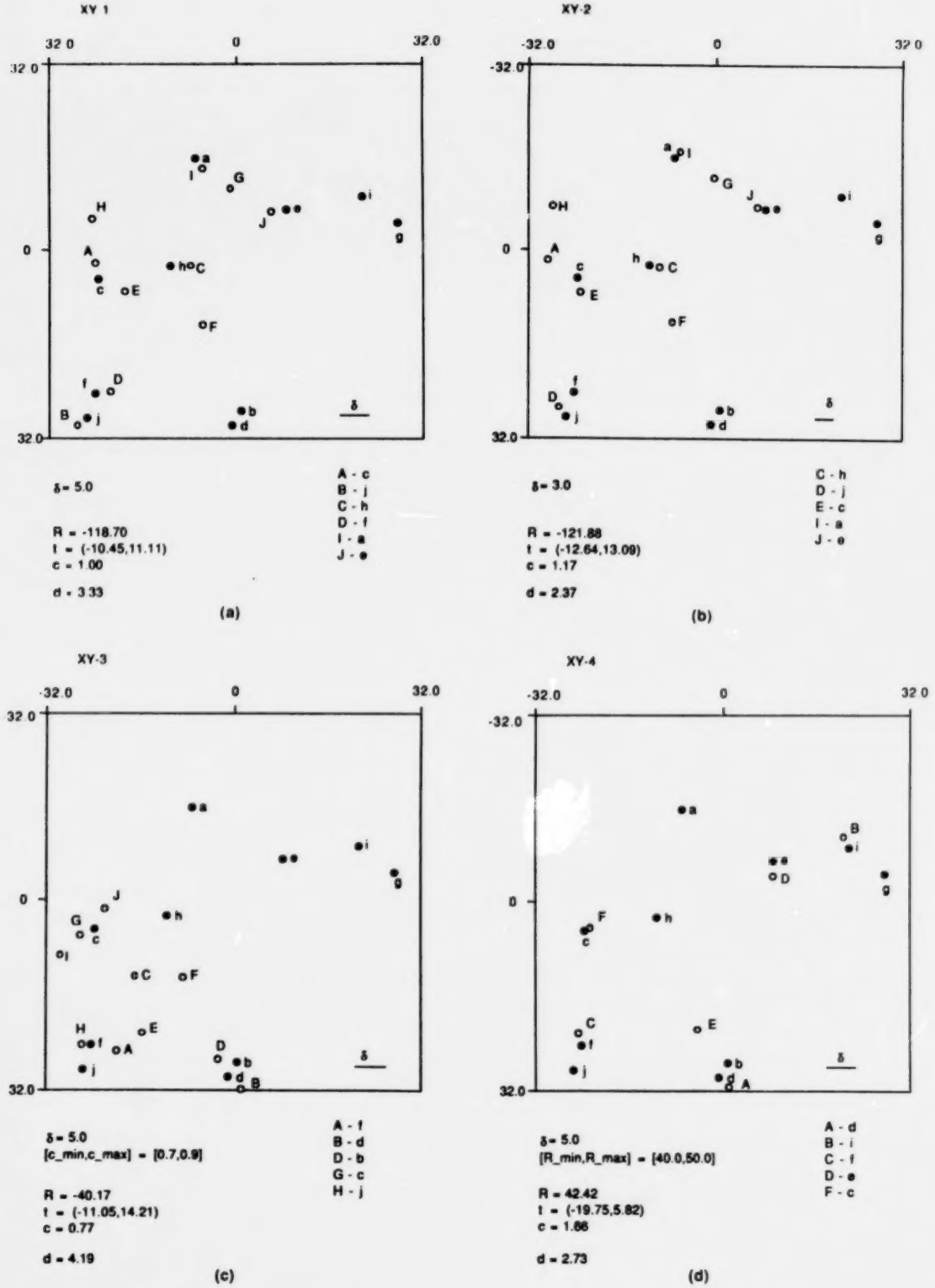


Figure 8. Matching results.

[c_min, c_max] = [0.7, 0.9], and in Fig. 7d, a range of an angle of a rotation was given, [R_min, R_max] = [30.0, 50.0] (degrees). The size of the obtained matching was 5 in both cases, and the number of obtained matchings was 4 and 15, respectively. The processing time for generating all the maximum matchings was 24.5 s for the case of (a), 18.3 s for (b), 3.2 s for (c), and 15.4 s for (d).

2.4.2 Point Pattern Matching in Three-dimensional Space

An example of point pattern matching in three-dimensional space is given in Figs. 9, 10, and 11. Figure 9 shows a projection of a randomly generated point pattern in 3-D space to the X-Y plane, and Fig. 10 shows a block whose vertices were used as a model pattern. No constraint is imposed on parameters of a registration, and δ is set to 5.0. The proposed matching algorithm generated four maximum matchings (because of the symmetry of the model object). One of the obtained matchings is shown in Fig. 11. The registration parameters of the obtained matching are as follows:

$$[27] \qquad R = \begin{bmatrix} 0.83 & -0.32 & 0.45 \\ -0.16 & -0.92 & -0.36 \\ 0.53 & 0.23 & -0.82 \end{bmatrix}, \quad t = \begin{bmatrix} 12.63 \\ 7.08 \\ -11.46 \end{bmatrix}, \quad c = 9.81$$

The maximum error distance of this matcning was 4.08. The processing time was 67.3 s.

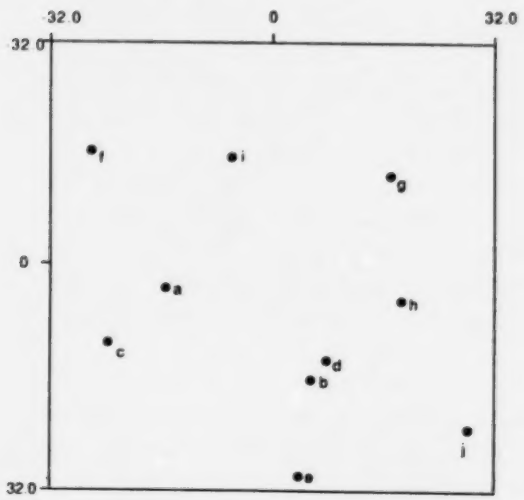


Figure 9. Randomly generated point patterns in 3-D space (projection to X-Y plane).

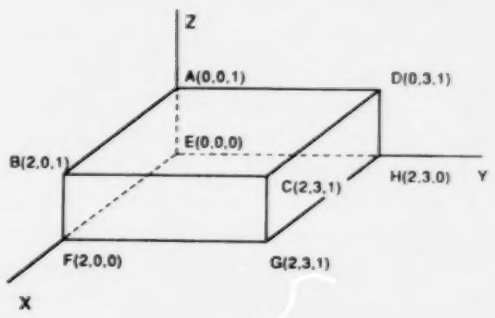


Figure 10. An object used as a model.

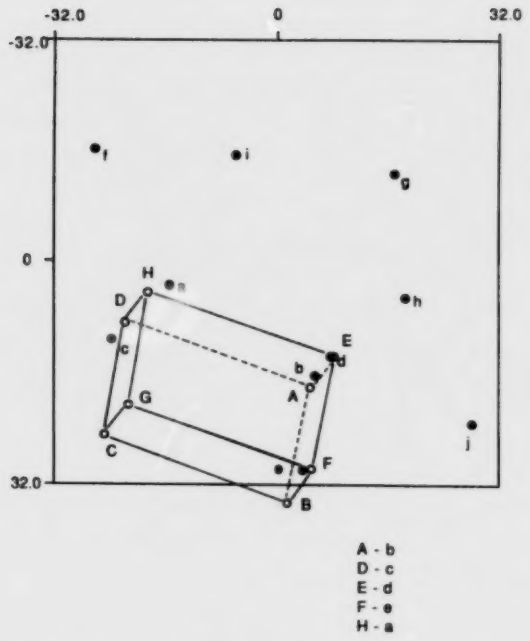


Figure 11. Matching result.

2.4.3 Shape Matching

The proposed point matching algorithm was applied to matching the shapes of machine parts when the parts are touching and/or overlapping each other. The model shape of each part is assumed to be extracted from a related image taken under similar conditions as the image of the objects. The image data were taken through a TV camera and digitized into 8-bit grey levels. The image size is 256 × 256 pixels. The grey level data were thresholded and converted into binary data using a discriminant thresholding method (10). Then the boundaries of the regions were extracted and approximated by straight-line segments based on an algorithm given in ref. 3. Figures 12 and 13 show the approximated boundaries of overlapping objects and models with the tolerance of 2.0 pixel length, respectively. A set of corner points of each boundary was used as the point pattern. In order to guide or prune the generation of partial matchings, several constraints were used.

1. Each corner was given a label numbered sequentially along a boundary. Since a matching must be consistent with respect to this labeling, we can avoid unnecessary searching by checking the consistency of the order of the labels. For example, when a partial matching $((c_1, c'_2), (c_2, c'_4), (c_3, c'_6))$ is given (see Fig. 14), a corner c_4 cannot

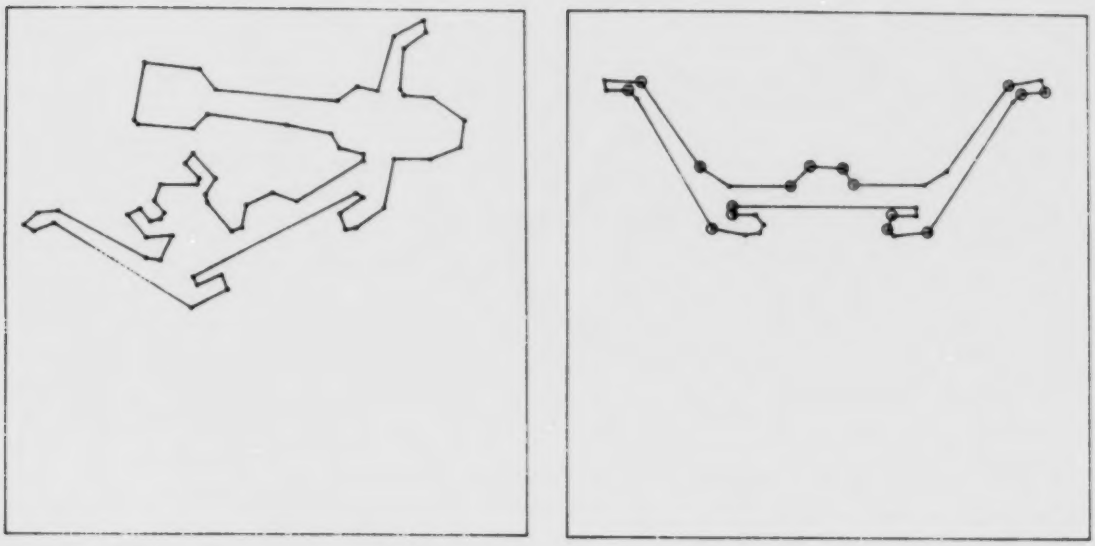
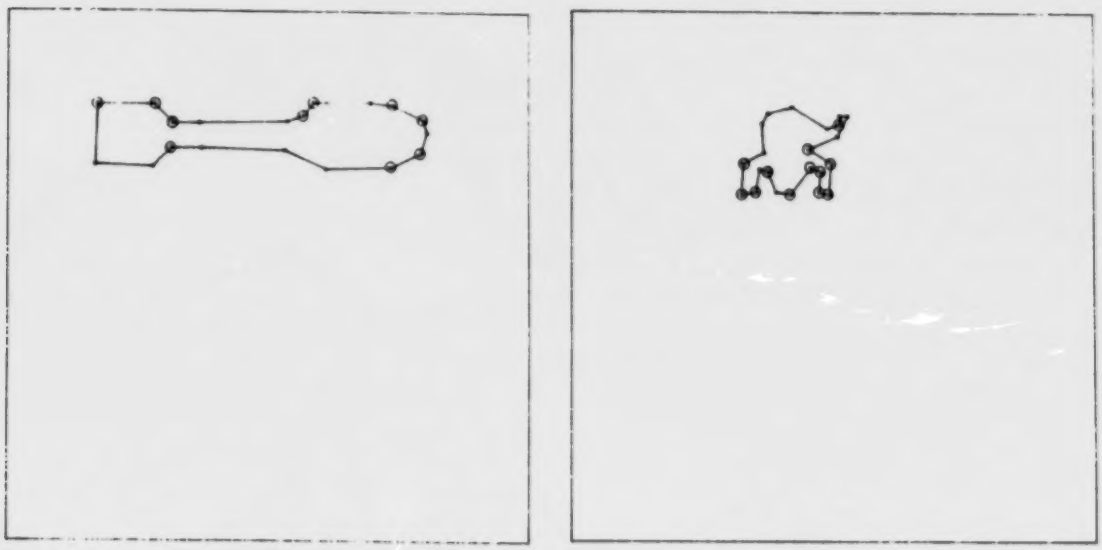


Figure 12. Polygonal approximation of a contour of overlapping objects.

(a)



(b)

(c)

Figure 13. Polygonal approximations of model contours.

correspond to c_3^1, since the matching becomes inconsistent in this case. Thus we can prune this matching without testing δ-boundedness.

2. An inner angle of each corner of a boundary was computed, and if the absolute difference between the inner angle of an object corner and that of a model corner was greater than a prescribed threshold value (TH_a = 20°), this object corner was not adopted as a corresponding corner of the model corner (Fig. 15).

3. A range for the scaling factor was given in advance, [c_min, c_max] = [0.9, 1.1].

4. The value δ was set to 6.0.

Figure 16 a-c shows the matching results between the shape in Fig. 12 and the shapes in Fig. 13. Similar to the previous example, only matchings which have the least maximum error distance among the obtained matchings are shown. The transformed shapes of Fig. 13 are superposed on the shape of Fig. 12. The circles in Fig. 13 indicate the corners which have the corresponding corners in the object image (i.e., the corners included in the obtained matching). Some of the parts have thickness, thus the contour shapes do not always produce unique corners because of lighting conditions. Despite this difficulty, all the objects and their positions were correctly identified. This shows the robust character of the proposed matching method. The number of obtained matchings was 4, 1, and 1, and the processing time of each matching was 62.9, 14.7, and 29.9 s, respectively, on a VAX-11/780.

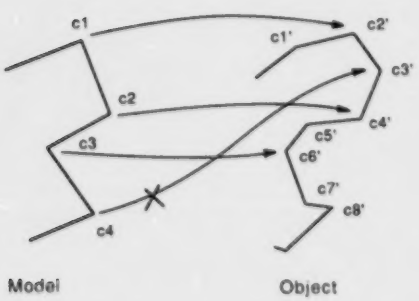


Figure 14. Inconsistent matching with respect to the order relation of corners.

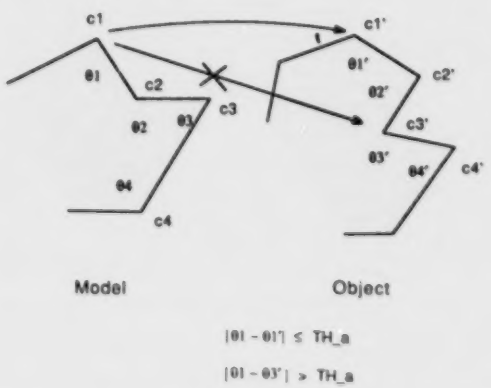


Figure 15. Inner angles at corners and a valid correspondence.

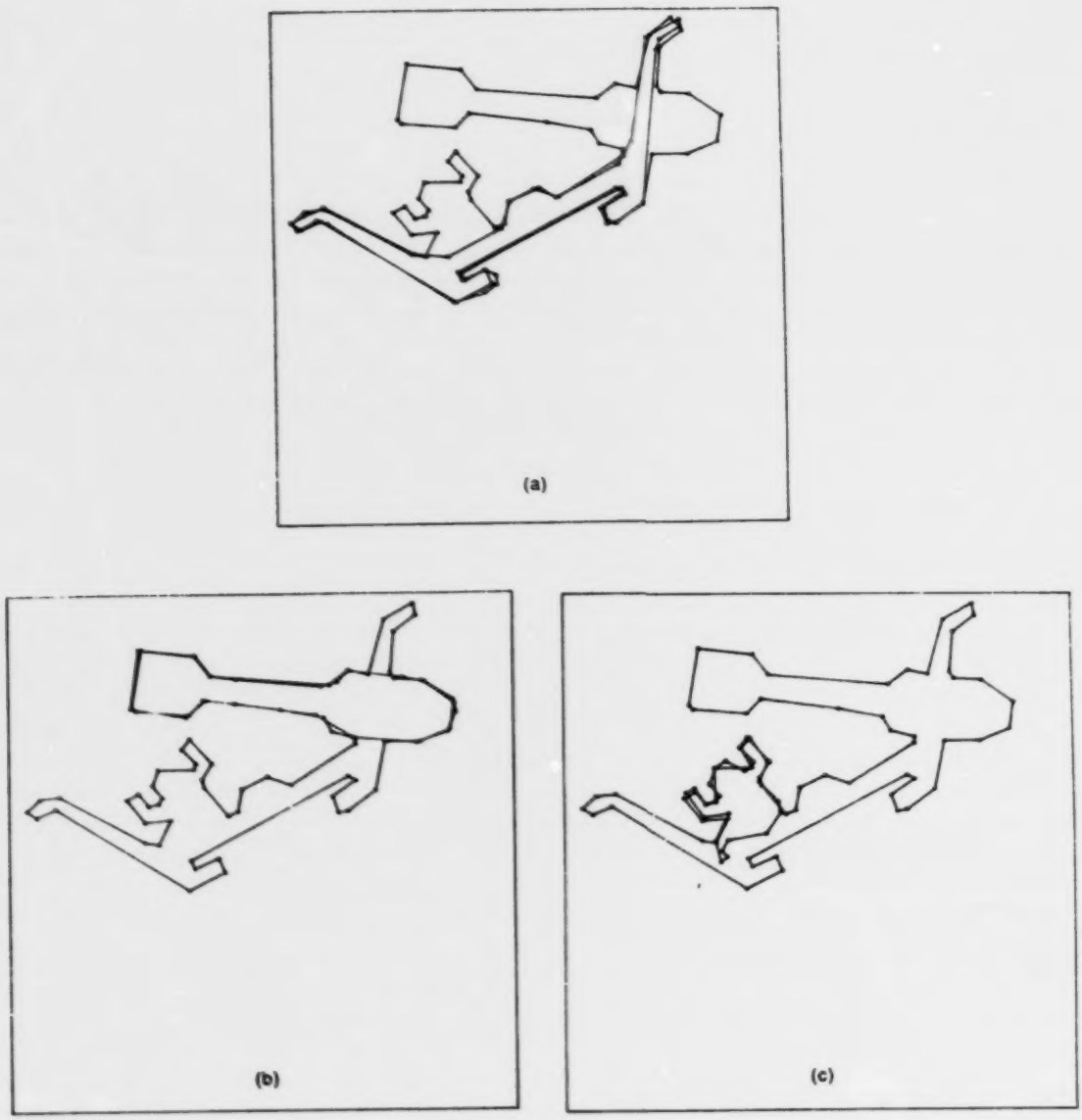


Figure 16. Matching results for shape matching.

3. PLANE PATTERN MATCHING

3.1 Definition of a Problem

In this section, we formalize a plane pattern matching problem in a similar way to a point pattern matching problem.

A plane is represented by a vector a and a scalar r. The equation of the plane p = (a, r) is

[28] $a'x - r = 0$

where a is the normal vector and r is the distance to the origin (see Fig. 17). A plane has two different equivalent definitions, (a, r) and (-a, -r). This orientation problem is easily solved by orienting the normal toward the outside of the object in the case of 3-D object recognition (11). A plane pattern P is a set of planes in m-dimensional Euclidean space. We write $P = \{p_1, p_2, \ldots\} = \{(a_1, r_1), (a_2, r_2), \ldots\}$. A matching of size n between two plane patterns, P and Q, is a one-to-one mapping f from a subset P_f of P of size n onto a subset Q_f of Q of the same size.

When a plane p = (a, r) is transformed by a registration T = (R, t, c), T(p) is given as follows:

[29] $T(p) = (Ra, cr + (Ra)'t)$

Thus, when two planes, p = (a, r) and q = (b, h), are given, the difference between q and the transformed plane T(p) is represented by two terms, $||b - Ra||$ and $|h - cr - (Ra)'t|$.

We say, similarly to point pattern matching, that a matching $f = ((p_1, q_1), (p_2, q_2), \ldots, (p_n, q_n))$ is δ-bounded if the maximum difference between the transformed planes and their corresponding planes is less than a prespecified tolerance measure, that is, f is called δ-bounded if the following holds,

[30a] $u = \max ||b_i - Ra_i|| < \delta_1$

[30b] $v = \max |h_i - cr_i - (Ra_i)'t| < \delta_2$

where $p_i = (a_i, r_i)$ and $q_i = (b_i, h_i)$, δ_1 and δ_2 are prespecified tolerances, and $T_f = (R, t, c)$ is an optimum registration from $P_f = (p_1, p_2,$

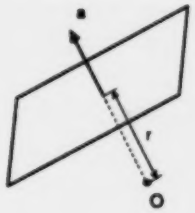


Figure 17. Representation of a plane.

..., p_n) to $Q_f = (q_1, q_2, \ldots, q_n)$ in the sense of the least-mean-squared error, i.e., T_f is a registration which minimizes the following error terms, e_1 and e_2, at the same time.

[31a] $$e_1 = (1/n) \sum_{i=1}^{n} ||b_i - Ra_i||^2$$

[31b] $$e_2 = (1/n) \sum_{i=1}^{n} |h_i - cr_i - (Ra_i)'t|^2$$

T_f can be determined since, as shown later, there exists T_f which minimizes e_1 and e_2 at the same time. We write ε_1 and ε_2 for the minimum values of e_1 and e_2.

Now a plane pattern matching problem between two plane patterns P and Q is defined as a problem of finding all δ-bounded matchings f of the maximum size N. N is the maximum number of planes that can be matched under the condition of δ-boundedness.

3.2 Plane Pattern Matching Algorithm

A plane pattern matching algorithm is formulated as a tree search. The basic tree search procedure is the same as that of a point pattern matching, and a search tree pruning method can be given similarly to the case of a point pattern matching problem.

Let f be a matching of size n. If the least-mean-squared error ε_1 of f defined by eq. [31a] is greater than δ_1^2, or ε_2 of f defined by eq. [31b] is greater than δ_2^2, then any matching g which includes f as a sub-matching cannot be δ-bounded. This property can be proved in a similar way to the case of a point pattern matching (see Section 2.2.2). Thus, if a partial matching satisfies this condition, the sub-tree with this partial matching as its root can be pruned and all matchings represented by this sub-tree will be discarded.

Thus, the method to compute the least-mean-squared errors, ε_1 and ε_2, of a given matching is a key issue also in plane pattern matching algorithm.

Theorem 2 gives these values.

Theorem 2: Let f be a matching of size n between plane patterns P and Q in m-dimensional space, and $P_f = (p_1, p_2, \ldots, p_n)$ and Q_f =

$(f(p_1), f(p_2), \ldots, f(p_n)) = (q_1, q_2, \ldots, q_n)$ $(p_i = (a_i, r_i)$ and $q_i = (b_i, h_i))$. Then the least-mean-squared errors ε_1 and ε_2 (the minimum values of e_1 and e_2 defined by eq. [31]) are given as follows:

[32a] $\varepsilon_1 = 2(1 - \mathrm{tr}(DS))$

[32b] $\varepsilon_2 = (1/n)(h'Gh - (h'Gr)^2/(r'Gr))$

where

[33a] $S_{ab} = (1/n) \sum_{i=1}^{n} b_i a_i'$

$= UDV'$: singular value decomposition of S_{ab}

$D = \mathrm{diag}(d_i)$, $d_1 > d_2 > \ldots > d_m > 0$

[33b] $S = \begin{cases} I & \text{if } \det(S_{ab}) > 0 \\ \mathrm{diag}(1, 1, \ldots, 1, -1) & \text{if } \det(S_{ab}) < 0 \end{cases}$

[33c] $r = (r_1, r_2, \ldots, r_n)'$

[33d] $h = (h_1, h_2, \ldots, h_n)'$

[33e] $G = I - A^+A$

[33f] $A = [a_1, a_2, \ldots, a_n]$

[33g] A^+ : pseudo inverse of A. $A^+ = A'(AA')^{-1}$

These minimum values are achieved when

[34a] $R = USV'$

[34b] $t = R(A')^+(h - cr)$

[34c] $c = (h'Gr)/(r'Gr)$

The proof of this theorem is given in Appendix B.

Obviously, S_{ab} can be computed recursively, thus the computation of ε_1 is easy if we store S_{ab} in each node. On the other hand, the possibility

of the recursive computation of ϵ_2 is not clear from the eq. [32b]. However, if we apply the matrix inversion lemma (12) to compute the pseudo inverse of A, δ_2 can be computed recursively. (We do not mention this derivation in this report. This will be described elsewhere.) Besides this, this computation is not difficult even if we compute it from scratch, since the computation of a pseudo inverse of A includes only a computation of an inverse matrix as shown in eq. [33g].

4. CONCLUSION

A new point and plane pattern matching algorithm formulated as a tree search procedure has been proposed. In order to prune the search tree, δ-boundedness was defined as a tolerance measure for matching, and a condition for rejecting non-δ-bounded partial matchings was given. A recursive computation method for this condition was also given. The proposed method is invariant to rotation, translation, and scaling, and applicable for geometric patterns in any dimensional space. It is easy to exploit global geometric constrains and additional information (such as semantic labels) within the same framework to guide the search. The performance of the algorithm in the case of point pattern matching is demonstrated by using randomly generated point patterns and shape matching problems of real scenes. These experiments show that the proposed algorithm can be one of the basic tools for many fields of image processing such as shape matching, motion analysis, stereoscopic vision system, and 3-D object recognition.

ACKNOWLEDGEMENTS

This work was done under the cooperative program between the Division of Electrical Engineering of the National Research Council Canada and the Electrotechnical Laboratory Japan. The authors are most grateful to the persons concerned of both organizations for making this work possible.

REFERENCES

1. A. Rosenfeld. Rapporteur for sensing systems. Workshop on the Research Needed to Advance the State of Knowledge in Robotics, Newport, RI. 1980. pp. 198-205.

2. H.S. Baird. Model-Based Image Matching Using Location. The MIT Press. 1985.

3. T. Pavlidis. Structural Pattern Recognition. Springer-Verlag, New York. 1977.

4. S. Ranade and A. Rosenfeld. Pattern matching by relaxation. Pattern Recognition, 12: 269-275; 1980.

5. D. Kahl, A. Rosenfeld, and A. Danker. Some experiments in point pattern matching. IEEE Trans. Syst., Man. Cybern. SMC-10: 105-116; 1980.

6. C.H. Papadimitriou and K. Steiglits. Combinatorial optimization: Algorithm and Complexity. Prentice-Hall Inc. 1982.

7. A.K.C. Wong. An algorithm for constellation matching. Proc. of the 8th International Conference of Pattern Recognition. 1986. Vol. 1, pp. 546-554.

8. L. Kitchen. Relaxation applied to matching quantitative relational structures. IEEE Trans. Syst., Man. Cybern. SMC-10: 96-101; 1980.

9. J.C. Simon, A. Checroun, and C. Roche. A method of comparing two patterns independent of possible transformations and small distortions. Pattern Recognition, 4: 73-81; 1972.

10. N. Otsu. A threshold selection method from grey-level histogram. IEEE Trans. Syst., Man Cybern. SMC-9: 62-66; 1979.

11. O.D. Faugeras and M. Hebert. The representation, recognition, and locating of 3-D objects. Int. J. Robot. Res. 5(3): 27-52; 1986.

12. B.O.D. Anderson and J.B. Moore. Optimal Filtering. Information and System Science Series. Prentice-Hall. 1979.

APPENDIX A. PROOF OF THEOREM 1

Before proving Theorem 1, we give a lemma.

Lemma: Let A and B be m * n matrices, R a m * m rotation matrix, and UDV' a singular value decomposition of AB' ($UU' = VV' = I$, $D = \mathrm{diag}(d_i)$, and $d_1 \geq d_2 \geq \ldots \geq d_m \geq 0$). Then the minimum value of $||A - RB||^2$ with respect to R is given as follows:

$$[A1] \qquad \min_R ||A - RB||^2 = ||A||^2 + ||B||^2 - 2\mathrm{tr}(DS)$$

where

$$S = \begin{cases} I & \text{if } \det(AB') > 0 \\ \mathrm{diag}(1, 1, \ldots, 1, -1) & \text{if } \det(AB') < 0 \end{cases}$$

When $\mathrm{rank}(AB') = m$, the optimum R which achieves the above minimum value is determined uniquely as follows:

$$[A2] \qquad R = USV'$$

Proof: Define the objective function F as

$$[A3] \qquad F = ||A - RB||^2 + \mathrm{tr}(L(R'R - I)) + l(\det(R) - 1)$$

where l is a Lagrange multiplier and L is a symmetric matrix of Lagrange multipliers. The second and third term of F represent the conditions for R to be an orthogonal and proper rotation matrix, respectively. Partial differentiations of F with respect to R, L, and l lead to the following system of equations (A1).

$$[A4a] \qquad \partial F/\partial R = -2AB' + 2RBB' + 2RL + lR = 0$$

$$[A4b] \qquad \partial F/\partial L = R'R - I = 0$$

$$[A4c] \qquad \partial F/\partial l = \det(R) - 1 = 0$$

where we used $(\partial/\partial R)\det(R) = \mathrm{adj}(R') = \det(R')(R')^{-1} = R$. ($\mathrm{adj}(R')$ is the adjoint matrix of R'.) From eq. [A4a] we have

$$[A5] \qquad RH = AB', \quad \text{where } H = BB' + L + (1/2)lI$$

By transposing both sides of the eq. [A5], we have the following (note that H is symmetric).

[A6] $HR' = BA'$

If we multiply each side of eq. [A5] by each side of eq. [A6] from the right, respectively, eq. [A7] is obtained since $R'R = I$.

[A7] $H^2 = BA'AB' = VD^2V'$

Obviously H and H^2 are commutative ($HH^2 = H^2H$), hence both can be reduced to diagonal forms by the same orthogonal matrix (A2). Thus, we can write

[A8] $H = VDSV'$, where $S = \text{diag}(s_i)$, $s_i = 1$ or -1

Certainly we have

[A9] $H^2 = VDSV'VDSV'$

$= VDSDSV' = VD^2S^2V' = VD^2V'$

Now from eq. [A8],

[A10] $\det(H) = \det(VDSV')$

$= \det(V)\det(D)\det(S)\det(V')$

$= \det(D)\det(S)$

On the other hand, from eq. [A5],

[A11] $\det(H) = \det(R'AB')$

$= \det(R')\det(AB')$

$= \det(AB')$

Thus,

[A12] $\det(D)\det(S) = \det(AB')$

Since singular values are non-negative, $\det(D) = d_1d_2\ldots d_m > 0$. Hence $\det(S)$ must be equal to 1 if $\det(AB') > 0$, -1 if $\det(AB') < 0$, and can be either if $\det(AB') = 0$.

Next, an extremum value of $||A - RB||^2$ is derived as follows: From eq. [A5], the following holds,

$$[A13] \quad ||A - RB||^2 = ||A||^2 + ||B||^2 - 2tr(AB'R')$$

$$= ||A||^2 + ||B||^2 - 2tr(R'AB')$$

$$= ||A||^2 + ||B||^2 - 2tr(H)$$

Substituting [A8] into [A13], we have

$$[A14] \quad || A - RB ||^2 = ||A||^2 + ||B||^2 - 2tr(VDSV')$$

$$= ||A||^2 + ||B||^2 - 2tr(DS)$$

$$= ||A||^2 + ||B||^2 - 2(d_1 s_1 + d_2 s_2 + \ldots + d_m s_m)$$

where we used $tr(VDSV') = tr(V'VDS) = tr(DS)$. Thus the minimum value of $||A - RB||^2$ is obviously achieved when $s_1 = s_2 = \ldots = s_m = 1$ if $\det(AB') > 0$, and $s_1 = s_2 = \ldots = s_{m-1} = 1$, $s_m = -1$ if $\det(AB') < 0$. This concludes the first half of the lemma.

Finally, if $\text{rank}(AB') = m$, H is a non-singular matrix, and thus it has its inverse $H^{-1} = (VDSV')^{-1} = VS^{-1}D^{-1}V' = VD^{-1}SV'$ (note $S^{-1} = S$ and $SD^{-1} = D^{-1}S$). Therefore, from eq. [A5] we have

$$[A15] \quad R = AB'H^{-1} = UDV'VD^{-1}DV' = USV'$$

This concludes the second half of Lemma. Q.E.D.

Now we give the proof of Theorem 1.

Proof: We represent the sets, X_f and Y_f, by m * n matrices $X = [x_1, x_2, \ldots, x_n]$ and $Y = [y_1, y_2, \ldots, y_n]$, respectively. Then e in

$$[A16] \quad e = (1/n) \sum_{x \in X_f} ||f(x) - T(x)||^2$$

is reformulated as follows:

$$[A17] \quad e = (1/n)||Y - cRX - th'||^2$$

where

$$[A18] \quad h = (1, 1, \ldots, 1)'$$

Here we introduce a n * n normalization matrix $K = I - (1/n)hh'$ ($K^2 = K' = K$). Using this matrix, the characteristics (s_x^2, s_y^2, S_{xy}) are written as follows:

[A19a] $$s_x^2 = (1/n)||XK||^2$$

[A19b] $$s_y^2 = (1/n)||YK||^2$$

[A19c] $$S_{xy} = (1/n)YKX'$$

Moreover, if we use the following equations,

[A20a] $$X = XK + (1/n)Xhh'$$

[A20b] $$Y = YK + (1/n)Yhh'$$

e is further reformulated as follows:

[A21] $$\begin{aligned} e &= (1/n)||YK + (1/n)Yhh' - cRXK - (c/n)RXhh' - th'||^2 \\ &= (1/n)||YK - cRXK + ((1/n)Yh - (c/n)RXh - t)h'||^2 \\ &= (1/n)||YK - cRXK - uh'||^2 \\ &= (1/n)(||YK - cRXK||^2 + ||uh'||^2 \\ &\qquad - 2tr(K(Y' - cX'R')uh')) \end{aligned}$$

where

[A22] $$u = -(1/n)Yh + (c/n)RXh + t$$

Since we can show eqs. [A23] and [A24],

[A23] $$\begin{aligned} tr(K(Y' - cX'R')uh') &= tr(h'(I - (1/n)hh')(Y' - cX'R')u) \\ &= tr((h' - h')(Y' - cX'R')u) \\ &= 0 \end{aligned}$$

[A24] $$||uh'||^2 = n||u||^2$$

e becomes

[A25] $$e = (1/n)||YK - cRXK||^2 + ||u||^2$$

From this equation, u must be equal to 0 in order to minimize e, that is,

[A26] $$t = (1/n)Yh - (c/n)RXh = m_y - cRm_x$$

since $(1/n)Yh = m_y$ and $(1/n)Xh = m_x$.

Next, when UDV' is the singular value decomposition of $S_{xy} = (1/n)YKX'$, the singular value decomposition of $YK(cXK)' = cYKK'X' = cYKX'$ is $cnUDV'$. Thus, the minimum value of $(1/n)||YK - cRXK||^2$ with respect to R is given in eq. [A27] from Lemma.

[A27] $$(1/n)(||YK||^2 + ||cXK||^2 - 2tr(cnDS)) = s_y^2 + c^2 s_x^2 - 2ctr(DS)$$

where

[A28] $$S = \begin{cases} I & \text{if } \det(S_{xy}) > 0 \\ \text{diag}(1, 1, \ldots, 1, -1) & \text{if } \det(S_{xy}) < 0 \end{cases}$$

Also from Lemma, if $\text{rank}(YKX') = \text{rank}(S_{xy}) = m$, the optimum R is

[A29] $$R = USV'$$

The minimum of eq. [A27] is obviously attained when

[A30] $$c = tr(DS)/s_x^2$$

Thus the minimum value ε of e is given as follows:

[A31] $$\varepsilon = s_y^2 + (tr(DS)/s_x^2)^2 s_x^2 - 2(tr(DS)/s_x^2)tr(DS)$$
$$= s_y^2 - (tr(DS))^2/s_x^2$$

This concludes the theorem.

REFERENCES

A1. M. Athans. The matrix minimum principle. Inf. Control, 11: 592-606; 1967.

A2. R. Bellman. Introduction to Matrix Analysis. McGraw-Hill. 1960.

APPENDIX B. PROOF OF THEOREM 2

Proof: Let $A = [a_1, a_2, \ldots, a_n]$, $B = [b_1, b_2, \ldots, b_n]$, $r = [r_1, r_2, \ldots, r_n)'$, and $h = (h_1, h_2, \ldots, h_n)'$. Then, the equations

$$[B1a] \qquad e_1 = (1/n) \sum_{i=1}^{n} ||b_i - Ra_i||^2$$

and

$$[B1b] \qquad e_2 = (1/n) \sum_{i=1}^{n} |h_i - cr_i - (Ra_i)'t|^2$$

are reformulated as follows:

$$[B2a] \qquad e_1 = (1/n)||B - RA||^2$$

$$[B2b] \qquad e_2 = (1/n)||h - cr - (RA)'t||^2$$

When R is given, the translation t which minimizes e_2 is

$$[B3] \qquad t = (A'R')^+(h - cr) = R(A')^+(h - cr)$$

where "+" indicates the pseudo inverse, and here we used the equation $(XR)^+ = R'X^+$ when R is orthogonal. Substituting [B3] in [B2b], we have

$$[B4] \qquad e_2 = (1/n)||h - cr - A'R'R(A')^+(h - cr)||^2$$

$$= (1/n)||(I - A'(A')^+)(h - cr)||^2$$

$$= (1/n)||(I - A^+A)(h - cr)||^2$$

where we used $A'(A')^+ = A'(A^+)' = (A^+A)' = A^+A$ (see ref. B1). Equation [B4] means that the minimum value of e_2 does not depend on R, i.e., it is constant with respect to R. Thus, we can minimizes e_1 in [B1a] and e_2 in [B4] independently; we need to minimize e_1 with respect to R, and e_2 with respect to c.

First, from Lemma in Appendix A, the minimum value ε_1 of e_1 is given as follows:

$$[B5] \qquad \varepsilon_1 = (1/n)(||A||^2 + ||B||^2 - 2\mathrm{tr}(nDS))$$

where we used $BA' = nS_{ab} = nUDV'$. S is defined in eq. [B6].

$$[B6] \qquad S = \begin{cases} I & \text{if } \det(S_{ab}) \geq 0 \\ \text{diag}(1, 1, \ldots, 1, -1) & \text{if } \det(S_{ab}) < 0 \end{cases}$$

Since the length of each normal vectors is 1, $||A||^2 = ||B||^2 = n$. Thus we have

$$[B7] \qquad \varepsilon_1 = (1/n)(n + n - 2\text{tr}(nDS))$$

$$= 2(1 - \text{tr}(DS))$$

This minimum value is achieved when $R = USV'$.

Next we minimize e_2. From eq. [B4], we have

$$[B8] \qquad e_2 = (1/n)(h - cr)'(I - A^+A)'(I - A^+A)(h - cr)$$

$$= (1/n)(h - cr)'(I - A^+A)(h - cr)$$

since $(I - A^+A)'(I - A^+A) = I - A^+A$ (see ref. B1). Thus,

$$[B9] \qquad e_2 = (1/n)(h'Gh - 2ch'Gr + c^2r'Gr)$$

where

$$G = I - A^+A$$

Hence, the minimum value ε_2 of e_2 is

$$[B10] \qquad \varepsilon_2 = (1/n)(h'Gh - (h'Gr)^2/(r'Gr))$$

This value is achieved when

$$[B11] \qquad c = h'Gr/(r'Gr).$$

This concludes the theorem.

REFERENCE

B1. A. Ben-Israel and T.N.E. Greville. Generalized Inverses: Theory and Application. A Wiley-International Publication. 1974.

APPENDIX C. Point Pattern Matching Program for 2-D

```
MODULE  PPM_2D(input,output);


type    arr2   = array[1..2[] of real;

{
    Explanation of arguments
 :: INPUT ::
    X  : List of model points.
    Y  : List of scene points.
    Nx : Number of model points.
    Ny : Number of scene points.
    MIRROR      : MIRROR=1 ==> det(R)=1 or -1, MIRROR<>1 ==> det(R)=1
    Delta       : Maximum distance of corresponding points.
    SW_c        : SW_c <> 1 ==> no constraint on c.
    min_c,max_c : Prespecified range of c.  0 <= min_c, max_c < 1e10.
    SW_Q        : SW_Q <> 1 ==> no constraint of Q.
    aprx_Q,TH_Q : Prespecified range of Q.  [aprx_Q-TH_Q,aprx_Q+TH_Q].
                  -180 <= aprx_Q <= 180.
  :: OUTPUT ::
    NS       : Number of obtained matchings.
    cpd      : Matching result (list of correspondences)
                 cpd[i,X_index] = Y_index.
    ev       : The least mean squared error.
    maxd     : Maximum distance.
    c        : Scaling.
    b        : Translation.
    Q        : Rotation (degrees).
                Rotation matrix = |   cos(Q)        sin(Q)     |
                                  | -detQ*sin(Q)  detQ*cos(Q)  |
    detQ     : det(Q).
    cpu      : cputime.


[GLOBAL] procedure PPM_2D
           (var X    :array[x1..x2 :integer] of arr2;
            var Y    :array[y1..y2 :integer] of arr2;
                Nx   :integer;
                Ny   :integer;
                MIRROR   :integer;
                Delta    :real;
                SW_c     :integer;
```

```
            min_c    :real;
            max_c    :real;
            SW_Q     :integer;
            aprx_Q   :real;
            TH_Q     :real;
        var NS   :integer;
        var cpd  :array[cpd1..cpd2 :integer; cpd3..cpd4 :integer]
                                                    of integer;
        var ev   :array[ev1..ev2 :integer] of real;
        var maxd :array[maxd1..maxd2 :integer] of real;
        var c    :array[c1..c2 :integer] of real;
        var b    :array[b1..b2 :integer] of arr2;
        var Q    :array[q1..q2 :integer] of real;
        var detQ :array[detq1..detq2 :integer] of integer;
        var cpu  :integer);


label 999;

const  maxN     = 20;

type   arr2_2     = array[1..2,1..2] of real;
       byte       = 0..255;
       ref_node   =  node;
       node       = record
                      sz      :integer;
                      p       :integer;
                      hx      :packed array[1..maxN] of byte;
                      hy      :set of byte;
                      cmin,cmax :real;
                      mx,my   :arr2;
                      xk,yk   :real;
                      ykx     :arr2_2;
                      ev      :real;
                      next    :ref_node;
                    end;

var    open_node  : ref_node;
       cand_node  : ref_node;
       garbage    : ref_node;
       SIZE       : integer;
       Delta2     : real;
       Gx,Gy      : array[1..maxN,1..maxN] of real;

       h1,h2,h3   :  integer;
```

```
{-- SVD ----------------------------------------}

{ Fortran subroutine in LINPACK.
  This subroutine must be linked when linking programs.
  Ex. $LINK main program,2dppm,linkpack/lib            }

[EXTERNAL] procedure ssvdc
               (%ref X    :array[x1..x2 :integer; x3..x4 :integer] of real;
                %ref LDX  :integer;
                %ref N    :integer;
                %ref P    :integer;
                %ref S    :array[s1..s2 :integer] of real;
                %ref E    :array[e1..e2 :integer] of real;
                %ref U    :array[u1..u2 :integer; u3..u4 :integer] of real;
                %ref LDU  :integer;
                %ref V    :array[v1..v2 :integer; v3..v4 :integer] of real;
                %ref LDV  :integer;
                %ref WORK :array[w1..w2 :integer] of real;
                %ref JOB  :integer;
                %ref INFO :integer); FORTRAN;


procedure svd_1(var A :arr2_2; var trd :real);
begin
 if MIRROR = 1 then
    trd:=sqrt(A[1,1]**2+A[1,2]**2+A[2,1]**2+A[2,2]**2
              +2*abs(A[1,1]*A[2,2]-A[1,2]*A[2,1]))
  else
    trd:=sqrt((A[1,1]+A[2,2])**2+(A[1,2]-A[2,1])**2)
end;


procedure svd_2(A :arr2_2; var Q :arr2_2; var  trd :real;
                           rank :integer);
 var  U,V         :arr2_2;
      D,E,WORK    :arr2;
      info,i,j,k :integer;
      r,r1,r2     :real;
begin
 r:=A[1,1]*A[2,2]-A[1,2]*A[2,1];
 ssvdc(A,2,2,2,D,E,U,2,V,2,WORK,11,info);
 if info <> 0 then writeln(' ERROR -- SVD ');
 if MIRROR = 1 then trd:=D[1]+D[2]
 else if rank = 2 then
```

```
  begin
   if r >= 0 then trd:=D[1]+D[2]
    else begin trd:=D[1]-D[2]; U[2,1]:=-U[2,1]; U[2,2]:=-U[2,2]; end;
  end else begin
   trd:=D[1];
   r1:=U[1,1]*U[2,2]-U[1,2]*U[2,1];
   r2:=V[1,1]*V[2,2]-V[1,2]*V[2,1];
   if r1*r2 < 0 then begin U[2,1]:=-U[2,1]; U[2,2]:=-U[2,2]; end;
  end;
 for i:=1 to 2 do for j:=1 to 2 do
   begin Q[i,j]:=0; for k:=1 to 2 do Q[i,j]:=Q[i,j]+V[k,i]*U[k,j]; end;
end;


{--- Miscellaneous ------------------------------}


function angle_Q(var Q :arr2_2):real;
 var  th,c,s  :real;
begin
 c:=Q[1,1]; s:=Q[1,2];
 if abs(c) < 1e-4 then
  begin if s >= 0 then th:=90 else th:=-90 end
  else begin
   th:=arctan(s/c)*180/3.1416;
   if c < 0 then
    if s >= 0 then th:=th+180 else th:=th-180;
  end;
 angle_Q:=th;
end;


procedure apply_Qbc(Q :arr2_2; b :arr2; c :real; x1 :arr2; var x2 :arr2);
 var  i,j  :integer;
begin
 for i:=1 to 2 do
  begin x2[i]:=b[i];
        for j:=1 to 2 do x2[i]:=x2[i]+c*Q[i,j]*x1[j]; end;
end;


procedure compute_graph;
 var  i,j  :integer;
begin
```

```
 for i:=1 to Nx-1 do for j:=i+1 to Nx do
  begin Gx[i,j]:=sqrt((X[i,1]-X[j,1])**2+(X[i,2]-X[j,2])**2);
        Gx[j,i]:=Gx[i,j]; end;
 for i:=1 to Ny-1 do for j:=i+1 to Ny do
  begin Gy[i,j]:=sqrt((Y[i,1]-Y[j,1])**2+(Y[i,2]-Y[j,2])**2);
        Gy[j,i]:=Gy[i,j]; end;
end;


procedure product_range(var c1,c2,c3,c4 :real);
 { [c1,c2] * [c3,c4] ==> [c1,c2] , if c2 < 0 then []*[] = null }
begin
 if c1 <= c3 then
  begin
   if c2 < c3 then c2:=-1
    else if c4 <= c2 then begin c1:=c3; c2:=c4; end
                     else c1:=c3;
  end else begin
   if c4 < c1 then c2:=-1
    else if c2 > c4 then c2:=c4;
  end;
end;


procedure disp_node(var t :ref_node);
 var  i,j  :integer;
begin
 with t  do
  begin
   writeln('----------------------');
   writeln(' size =',sz:4,'   ev =',ev:7:2);
   write('    p =',p:3,' :: ');
   for i:=1 to p do write(hx[i]:3); writeln;
   for i:=1 to Ny do if (i in hy) then write(i:3); writeln;
   writeln(' cmin =',cmin:7:2,'    cmax =',cmax:7:2);
   writeln(' xk =',xk:7:2,'     yk =',yk:7:2);
   writeln(' mx =',mx[1]:7:2,mx[2]:7:2,'    my =',my[1]:7:2,my[2]:7:2);
   write(' ykx =');
   for i:=1 to 2 do for j:=1 to 2 do write(ykx[i,j]:7:2); writeln;
  end;
end;
```

```
{--- Node list handling  ----------------------}


function  new_node:ref_node;
 var t  :ref_node;
begin
 if garbage = nil then begin new(t); t .next:=nil; new_node:=t; end
  else begin t:=garbage; garbage:=garbage .next;
             t .next:=nil; new_node:=t;      end;
end;


procedure dispose_node(t :ref_node);
begin
 t .next:=garbage; garbage:=t;
end;


procedure insert_node(var t :ref_node);
begin
 t .next:=open_node;
 open_node:=t;
end;


procedure init_node_list;
 var  t   :ref_node;
      i,j :integer;
begin
 cand_node:=nil; garbage:=nil; open_node:=nil;
 new(t);
 with t  do
  begin
   sz:=0; p:=0;
   hy:=[]; for i:=1 to Ny do hy:=hy+[i];
   if SW_c = 1 then begin cmin:=min_c; cmax:=max_c; end
    else begin cmin:=0; cmax:=lel0; end;
   mx[1]:=0; mx[2]:=0; my[1]:=0; my[2]:=0; xk:=0; yk:=0;
   for i:=1 to 2 do for j:=1 to 2 do ykx[i,j]:=0;
   ev:=0; next:=nil;
  end;
 open_node:=t;
end;
```

```
{--- Update and check ---------------------------}


function update_and_check(t1,t2 :ref_node):boolean;
 label 10,20;
 var  i,j,k,k1,k2,k3  :integer;
      c,d,r           :real;
      xx,yy,b         :arr2;
      Q               :arr2_2;
      f               :boolean;

begin
 h1:=h1+1;
 if t1 .sz = 0 then
  begin
   k:=t2 .p;
   t2 .mx:=X[k]; t2 .my:=Y[t2 .hx[k]];
   t2 .xk:=0; t2 .yk:=0; t2 .ykx:=t1 .ykx; t2 .ev:=0;
   f:=true;
  end else begin
   k1:=t2 .sz-1; k2:=t2 .p; k3:=t2 .hx[k2];
   for i:=1 to 2 do
    begin
     t2 .mx[i]:=(t1 .mx[i]*k1+X[k2,i])/(k1+1);
     t2 .my[i]:=(t1 .my[i]*k1+Y[k3,i])/(k1+1);
     xx[i]:=t1 .mx[i]-X[k2,i];
     yy[i]:=t1 .my[i]-Y[k3,i];
    end;
   r:=xx[1]**2+xx[2]**2;
   t2 .xk:=t1 .xk+r*k1/(k1+1);
   r:=yy[1]**2+yy[2]**2;
   t2 .yk:=t1 .yk+r*k1/(k1+1);
   for i:=1 to 2 do for j:=1 to 2 do
     t2 .ykx[i,j]:=t1 .ykx[i,j]+yy[i]*xx[j]*k1/(k1+1);
   if SW_Q <> 1 then
    begin
     f:=true;
     svd_1(t2 .ykx,d);
     t2 .ev:=(t2 .yk-(d**2)/t2 .xk)/(k1+1);
     if t2 .ev > Delta2 then begin f:=false; goto 10; end;
     if SW_c = 1 then
      begin c:=d/t2 .xk;
            if (c < t2 .cmin) or (c > t2 .cmax) then f:=false; end;
     10:
    end else begin
```

```
      f:=true;
      if kl =1 then svd_2(t2 .ykx,Q,d,1) else svd_2(t2 .ykx,Q,d,2);
      t2 .ev:=(t2 .yk-(d**2)/t2 .xk)/(kl+1);
      if t2 .ev > Delta2 then begin f:=false; goto 20; end;
      if SW_c = 1 then
       begin c:=d/t2 .xk;
             if (c < t2 .cmin) or (c > t2 .cmax)
               then begin f:=false; goto 20; end;   end;
      r:=angle_Q(Q);
      r:=abs(r-aprx_Q); if r > 180 then r:=360-r;
      if r > TH_Q then f:=false;
      20:
     end;
   end;
  update_and_check:=f;
end;


function final_check(t :ref_node; var maxd :real;
                     var Q :arr2_2; var b :arr2; var c :real):boolean;
 label 10;
 var   xx    :arr2;
       d,r   :real;
       i,j,k   :integer;
begin
 h2:=h2+1;
 maxd:=0;
 svd_2(t .ykx,Q,d,2);
 c:=d/t .xk;
 for i:=1 to 2 do
  begin b[i]:=t .my[i];
        for j:=1 to 2 do b[i]:=b[i]-c*Q[i,j]*t .mx[j]; end;
 for i:=1 to t .p do
  begin
   k:=t .hx[i];
   if k <> 0 then
    begin
     apply_Qbc(Q,b,c,X[i],xx);
     r:=(xx[1]-Y[k,1])**2+(xx[2]-Y[k,2])**2;
     if r > Delta2 then begin final_check:=false; goto 10 end;
     if r > maxd then maxd:=r;
    end;
  end;
 final_check:=true;
 10:
end;
```

```
procedure c_range_check(t :ref_node; ix,iy :integer; var c1,c2 :real);
 label 10;
 var  c3,c4  :real;
      i      :integer;

begin
 c1:=t .cmin; c2:=t .cmax;
 if t .sz <> 0 then
  for i:=1 to t .p do if t .hx[i] <> 0 then
   begin
    c3:=(Gy[t .hx[i],iy]-2*Delta)/Gx[i,ix];
    c4:=(Gy[t .hx[i],iy]+2*Delta)/Gx[i,ix];
    if c3 < 0 then c3:=0;
    product_range(c1,c2,c3,c4);
    if c2 < 0 then goto 10;
   end;
 10:
end;


{--- Found ------------------------------}


procedure found;
 var  QQ     :arr2_2;
      i      :integer;
      d,cc   :real;
      bb     :arr2;
      t,tt   :ref_node;

     procedure insert(tt :ref_node; var t :ref_node);
     begin
      if t = nil then t:=tt
       else if tt .yk < t .yk then begin tt .next:=t; t:=tt; end
       else insert(tt,t .next);
     end;

     procedure sort_solutions(var t :ref_node);
      var  tt  :ref_node;
     begin
      if t <> nil then
       begin
        tt:=t; t:=t .next; tt .next:=nil;
        sort_solutions(t);
```

```
        insert(tt,t);
       end;
     end;

begin
 tt:=nil;
 while cand_node <> nil do
  begin
   t:=cand_node; cand_node:=cand_node .next;
   if final_check(t,d,QQ,bb,cc) then
    begin t .ykx:=QQ; t .mx:=bb; t .xk:=cc; t .yk:=sqrt(d);
          t .next:=tt; tt:=t; end;
  end;
 sort_solutions(tt);
 NS:=0;
 while tt <> nil do
  begin
   NS:=NS+1;
   t:=tt; tt:=tt .next;
   for i:=1 to t .p do cpd[NS,i]:=t .hx[i];
   for i:=t .p+1 to Nx do cpd[Ns,i]:=0;
   ev[NS]:=t .ev;
   maxd[NS]:=t .yk;
   c[NS]:=t .xk;
   b[NS]:=t .mx;
   with t  do d:=ykx[1,1]*ykx[2,2]-ykx[1,2]*ykx[2,1];
   Q[NS]:=angle_Q(t .ykx);
   if d > 0 then detQ[NS]:=1 else detQ[NS]:=-1;
  end;
end;


{--- Node Expansion ----------------------------}


procedure expand;
 label   10;
 var  Q          :arr2_2;
      b          :arr2;
      c,d,c1,c2  :real;
      t1,t2,t3   :ref_node;
      i,j,il     :integer;
begin
 if open_node = nil then goto 999
  else begin t1:=open_node; open_node:=open_node .next; end;
```

```
il:=tl .p+1;
if il <= Nx then
 begin
  if (Nx-tl .p+tl .sz) > SIZE then
   begin
    t2:=new_node;
    t2 :=tl ;
    t2 .p:=il; t2 .hx[il]:=0;
    t2 .next:=nil;
    insert_node(t2);
   end;
  for i:=Ny downto 1 do if (i in tl .hy) then
   begin
    h3:=h3+1;
    c_range_check(tl,il,i,cl,c2);
    if c2 > 0 then
     begin
      t2:=new_node;
      t2 .sz:=tl .sz+1;
      t2 .p:=il;
      for j:=1 to tl .p do t2 .hx[j]:=tl .hx[j];
      t2 .hx[il]:=i;
      t2 .hy:=tl .hy-[i];
      t2 .cmin:=cl; t2 .cmax:=c2;
      if update_and_check(tl,t2) = false then dispose_node(t2)
       else if t2 .sz < SIZE then insert_node(t2)
       else if t2 .sz = SIZE then
        begin t3:=new_node; t3 :=t2 ;
              t3 .next:=cand_node; cand_node:=t3; insert_node(t2); end
       else begin
        if final_check(t2,d,Q,b,c) = false then goto 10;
        while cand_node <> nil do
         begin t3:=cand_node; cand_node:=cand_node .next;
                                          dispose_node(t3); end;
         cand_node:=new_node; cand_node :=t2 ;
         SIZE:=SIZE+1; writeln(SIZE);
         10:
         insert_node(t2);
        end;
      end;
    end;
   end;
  dispose_node(tl);
 end;
```

```
{--- Main procedure ----------------------------------------}


begin
 cpu:=clock; h1:=0; h2:=0; h3:=0;
 SIZE:=2; Delta2:=Delta*Delta;
 compute_graph;
 init_node_list;
 while true do expand;
 999:
 writeln('Finished');
 found;
 cpu:=clock-cpu;
 writeln('time =',cpu,'    check =',h3,'   update =',h1,'   final =',h2);
 writeln;
end;


end.
```

APPENDIX D. Point Pattern Matching Program for 3-D

```
MODULE  PPM_3D(input,output);


type    arr3   = array[1..3] of real;
        arr3_3 = array[1..3,1..3] of real;


{
 ###Explanation of arguments##############################################
 :: INPUT ::
    X  : List of model points.
    Y  : List of scene points.
    Nx : Number of model points.
    Ny : Number of scene points.
    Delta         : Maximum distance of corresponding points.
    SW_c          : SW_c <> 1 ==> no constraint on c.
    min_c,max_c   : Prespecified range of c.  0 <= min_c, max_c < 1e10.
    SW_Q          : SW_Q <> 1 ==> no constraint on Q.
    aprx_Q,TH_Q   : Prespecified range of Q.
 :: OUTPUT ::
    NS        : Number of obtained solutions.
    cpd       : Matching result (list of correspondences)
                   cpd[i,X_index] = Y_index.
    ev        : The least mean squared error.
    maxd      : Maximum distance.
    c         : Scaling.
    b         : Translation.
    Q         : Rotation matrix.
    cpu       : cpu time.
 }


 [GLOBAL] procedure PPM_3D
             (var X    :array[x1..x2 :integer] of arr3;
              var Y    :array[y1..y2 :integer] of arr3;
                  Nx   :integer;
                  Ny   :integer;
                  Delta    :real;
                  SW_c     :integer;
                  min_c    :real;
                  max_c    :real;
                  SW_Q     :integer;
```

```
            aprx_Q   :arr3_3;
            TH_Q     :real;
        var NS   :integer;
        var cpd  :array[cpd1..cpd2 :integer; cpd3..cpd4 :integer]
                                                    of integer;
        var ev   :array[ev1..ev2 :integer] of real;
        var maxd :array[maxd1..maxd2 :integer] of real;
        var c    :array[c1..c2 :integer] of real;
        var b    :array[b1..b2 :integer] of arr3;
        var Q    :array[q1..q2 :integer] of arr3_3;
        var cpu  :integer);

label 999;

const  maxN     = 20;

type   byte       = 0..255;
       ref_node   =  node;
       node       = record
                      sz      :integer;
                      p       :integer;
                      hx      :packed array[1..maxN] of byte;
                      hy      :set of byte;
                      cmin,cmax  :real;
                      mx,my   :arr3;
                      xk,yk   :real;
                      ykx     :arr3_3;
                      ev      :real;
                      next    :ref_node;
                    end;

var    open_node  : ref_node;
       cand_node  : ref_node;
       garbage    : ref_node;
       SIZE       : integer;
       Delta2     : real;
       Gx,Gy      : array[1..maxN,1..maxN] of real;

       h1,h2      : integer;
```

```
{-- SVD ----------------------------------------}

{ Fortran subroutine in LINPACK.
  This subroutine must be linked when linking programs.
  Ex. $LINK main_program,3dppm,linkpack/lib            }

[EXTERNAL] procedure ssvdc
               (%ref X    :array[x1..x2 :integer; x3..x4 :integer] of real;
                %ref LDX  :integer;
                %ref N    :integer;
                %ref P    :integer;
                %ref S    :array[s1..s2 :integer] of real;
                %ref E    :array[e1..e2 :integer] of real;
                %ref U    :array[u1..u2 :integer; u3..u4 :integer] of real;
                %ref LDU  :integer;
                %ref V    :array[v1..v2 :integer; v3..v4 :integer] of real;
                %ref LDV  :integer;
                %ref WORK :array[w1..w2 :integer] of real;
                %ref JOB  :integer;
                %ref INFO :integer);FORTRAN;


function det(var A :arr3_3):real;
begin
 det:=A[1,1]*A[2,2]*A[3,3]+A[2,1]*A[3,2]*A[1,3]+A[3,1]*A[2,3]*A[1,2]
    -A[1,1]*A[3,2]*A[2,3]-A[1,2]*A[2,1]*A[3,3]-A[1,3]*A[2,2]*A[3,1];
end;


procedure svd_1(A :arr3_3; var trd :real);
 var  U,V       :arr3_3;
      D,E,WORK  :arr3;
      info      :integer;
      r         :real;
begin
 r:=det(A);
 ssvdc(A,3,3,3,D,E,U,3,V,3,WORK,00,info);
 if info <> 0 then writeln(' ERROR -- SVD ');
 if r >= 0 then trd:=D[1]+D[2]+D[3] else trd:=D[1]+D[2]-D[3];
end;


procedure svd_2(A :arr3_3; var Q :arr3_3; var  trd :real;
                       rank :integer);
 var  U,V        :arr3_3;
```

```
      D,E,WORK   :arr3;
      info,i,j,k :integer;
      r,r1,r2    :real;
begin
 r:=det(A);
 ssvdc(A,3,3,3,D,E,U,3,V,3,WORK,11,info);
 if info <> 0 then writeln(' ERROR -- SVD ');
 case rank of
  3 : begin
       if r >= 0 then trd:=D[1]+D[2]+D[3]
        else begin trd:=D[1]+D[2]-D[3];
                   U[3,1]:=-U[3,1]; U[3,2]:=-U[3,2]; U[3,3]:=-U[3,3]; end;
      end;
  2 : begin
       trd:=D[1]+D[2];
       r1:=det(U);
       r2:=det(V);
       if r1*r2 < 0 then
        begin U[3,1]:=-U[3,1]; U[3,2]:=-U[3,2]; U[3,3]:=-U[3,3] end;
      end;
  otherwise writeln(' ERROR - ');
 end;
 for i:=1 to 3 do for j:=1 to 3 do
   begin Q[i,j]:=0; for k:=1 to 3 do Q[i,j]:=Q[i,j]+V[k,i]*U[k,j]; end;
end;


{--- Miscellaneous ------------------------------}

procedure apply_Qbc(Q :arr3_3; b :arr3; c :real; x1 :arr3; var x2 :arr3);
 var  i,j  :integer;
begin
 for i:=1 to 3 do
  begin x2[i]:=b[i];
        for j:=1 to 3 do x2[i]:=x2[i]+c*Q[i,j]*x1[j]; end;
end;


procedure compute_graph;
 var  i,j  :integer;
begin
 for i:=1 to Nx-1 do for j:=i+1 to Nx do
  begin Gx[i,j]:=sqrt((X[i,1]-X[j,1])**2+(X[i,2]-X[j,2])**2
                                        +(X[i,3]-X[j,3])**2);
        Gx[j,i]:=Gx[i,j]; end;
 for i:=1 to Ny-1 do for j:=i+1 to Ny do
```

```
  begin Gy[i,j]:=sqrt((Y[i,1]-Y[j,1])**2+(Y[i,2]-Y[j,2])**2
                                                +(Y[i,3]-Y[j,3])**2);
        Gy[j,i]:=Gy[i,j]; end;
end;


procedure product_range(var c1,c2,c3,c4 :real);
 { [c1,c2] * [c3,c4] ==> [c1,c2] , if c2 < 0 then []*[] = null }
begin
 if c1 <= c3 then
  begin
   if c2 < c3 then c2:=-1
    else if c4 <= c2 then begin c1:=c3; c2:=c4; end
                     else c1:=c3;
  end else begin
   if c4 < c1 then c2:=-1
    else if c2 > c4 then c2:=c4;
  end;
end;


procedure disp_node(var t :ref_node);
 var  i,j  :integer;
begin
 with t  do
  begin
   writeln('----------------------');
   writeln(' size =',sz:4,'   ev =',ev:7:2);
   write('    p =',p:3,' :: ');
   for i:=1 to p do write(hx[i]:3); writeln;
   for i:=1 to Ny do if (i in hy) then write(i:3); writeln;
   writeln(' cmin =',cmin:7:2,'    cmax =',cmax:7:2);
   writeln(' xk =',xk:7:2,'     yk =',yk:7:2);
   writeln(' mx =',mx[1]:7:2,mx[2]:7:2,mx[3]:7:2,
           '    my =',my[1]:7:2,my[2]:7:2,my[3]:7:2);
   write(' ykx =');
   for i:=1 to 3 do for j:=1 to 3 do write(ykx[i,j]:7:2); writeln;
  end;
end;


{--- Node list handling  ----------------------}


function  new_node:ref_node;
 var t  :ref_node;
```

```
begin
 if garbage = nil then begin new(t); t .next:=nil; new_node:=t; end
  else begin t:=garbage; garbage:=garbage .next;
             t .next:=nil; new_node:=t;      end;
end;


procedure dispose_node(t :ref_node);
begin
 t .next:=garbage; garbage:=t;
end;


procedure insert_node(var t :ref_node);
begin
 t .next:=open_node;
 open_node:=t;
end;


procedure init_node_list;
 var  t   :ref_node;
      i,j :integer;
begin
 cand_node:=nil; garbage:=nil; open_node:=nil;
 new(t);
 with t  do
  begin
   sz:=0; p:=0;
   hy:=[]; for i:=1 to Ny do hy:=hy+[i];
   if SW_c = 1 then begin cmin:=min_c; cmax:=max_c; end
    else begin cmin:=0; cmax:=lel0; end;
   for i:=1 to 3 do begin mx[i]:=0; my[i]:=0; end;
   for i:=1 to 3 do for j:=1 to 3 do ykx[i,j]:=0;
   ev:=0; next:=nil;
  end;
 open_node:=t;
end;


{--- Update and check ------------------------}


function update_and_check(t1,t2 :ref_node):boolean;
 label 10,20,30;
```

```
var  i,j,k,kl,k2,k3  :integer;
     c,d,r           :real;
     xx,yy,b         :arr3;
     Q               :arr3_3;
     f               :boolean;

    procedure upd;
    begin
     kl:=t2 .sz-1; k2:=t2 .p; k3:=t2 .hx[k2];
     for i:=1 to 3 do
      begin
       t2 .mx[i]:=(t1 .mx[i]*kl+X[k2,i])/(kl+1);
       t2 .my[i]:=(t1 .my[i]*kl+Y[k3,i])/(kl+1);
       xx[i]:=t1 .mx[i]-X[k2,i];
       yy[i]:=t1 .my[i]-Y[k3,i];
      end;
     r:=xx[1]**2+xx[2]**2+xx[3]**2;
     t2 .xk:=t1 .xk+r*kl/(kl+1);
     r:=yy[1]**2+yy[2]**2+yy[3]**2;
     t2 .yk:=t1 .yk+r*kl/(kl+1);
     for i:=1 to 3 do for j:=1 to 3 do
       t2 .ykx[i,j]:=t1 .ykx[i,j]+yy[i]*xx[j]*kl/(kl+1);
   end;

begin
 hl:=hl+1;
 case t2 .sz of
  1 : begin
       k:=t2 .p;
       t2 .mx:=X[k]; t2 .my:=Y[t2 .hx[k]];
       t2 .xk:=0; t2 .yk:=0; t2 .ykx:=t1 .ykx; t2 .ev:=0;
       f:=true;
      end;
  2 : begin
       upd;
       f:=true;
       svd_1(t2 .ykx,d);
       t2 .ev:=(t2 .yk-(d**2)/t2 .xk)/(kl+1);
       if t2 .ev > Delta2 then begin f:=false; goto 30; end;
       if SW_c = 1 then
        begin c:=d/t2 .xk;
              if (c < t2 .cmin) or (c > t2 .cmax) then f:=false; end;
       30:
      end;
```

```
  otherwise begin
        upd;
        if SW_Q <> 1 then
         begin
          f:=true;
          svd_1(t2 .ykx,d);
          t2 .ev:=(t2 .yk-(d**2)/t2 .xk)/(k1+1);
          if t2 .ev > Delta2 then begin f:=false; goto 10; end;
          if SW_c = 1 then
           begin c:=d/t2 .xk;
                 if (c < t2 .cmin) or (c > t2 .cmax) then f:=false; end;
          10:
         end else begin
          f:=true;
          if k1 = 2 then svd_2(t2 .ykx,Q,d,2) else svd_2(t2 .ykx,Q,d,3);
          t2 .ev:=(t2 .yk-(d**2)/t2 .xk)/(k1+1);
          if t2 .ev > Delta2 then begin f:=false; goto 20; end;
          if SW_c = 1 then
           begin
            c:=d/t2 .xk;
            if (c < t2 .cmin) or (c > t2 .cmax) then
              begin f:=false; goto 20; end;
           end;
          r:=0; for j:=1 to 3 do for k:=1 to 3 do r:=r+Q[j,k]*aprx_Q[j,k];
          r:=2*(3-r);
          if r > TH_Q then f:=false;
          20:
         end;
       end;
 end;
 update_and_check:=f;
end;


function final_check(t :ref_node; var maxd :real;
                     var Q :arr3_3; var b :arr3; var c :real):boolean;
 label 10;
 var   xx    :arr3;
       d,r   :real;
       i,j,k   :integer;
begin
 h2:=h2+1;
 maxd:=0;
 svd_2(t .ykx,Q,d,3);
 c:=d/t .xk;
```

```
for i:=1 to 3 do
 begin b[i]:=t .my[i];
       for j:=1 to 3 do b[i]:=b[i]-c*Q[i,j]*t .mx[j]; end;
for i:=1 to t .p do
 begin
  k:=t .hx[i];
  if k <> 0 then
   begin
    apply_Qbc(Q,b,c,X[i],xx);
    r:=0; for j:=1 to 3 do r:=r+(xx[j]-Y[k,j])**2;
    if r > Delta2 then begin final_check:=false; goto 10 end;
    if r > maxd then maxd:=r;
   end;
 end;
final_check:=true;
10:
end;


procedure c_range_check(t :ref_node; ix,iy :integer; var c1,c2 :real);
 label 10;
 var  c3,c4  :real;
      i      :integer;
begin
 c1:=t .cmin; c2:=t .cmax;
 if t .sz <> 0 then
  for i:=1 to t .p do if t .hx[i] <> 0 then
   begin
    c3:=(Gy[t .hx[i],iy]-2*Delta)/Gx[i,ix];
    c4:=(Gy[t .hx[i],iy]+2*Delta)/Gx[i,ix];
    if c3 < 0 then c3:=0;
    product_range(c1,c2,c3,c4);
    if c2 < 0 then goto 10;
   end;
 10:
end;


{--- Found ----------------------------}


procedure found;
 var  QQ   :arr3_3;
      bb   :arr3;
      d,cc :real;
```

```
      i    :integer;
      t,tt :ref_node;

     procedure insert(tt :ref_node; var t :ref_node);
     begin
      if t = nil then t:=tt
       else if tt .yk < t .yk then begin tt .next:=t; t:=tt; end
       else insert(tt,t .next);
     end;

     procedure sort_solutions(var t :ref_node);
      var  tt  :ref_node;
     begin
      if t <> nil then
       begin
        tt:=t; t:=t .next; tt .next:=nil;
        sort_solutions(t);
        insert(tt,t);
       end;
     end;

begin
 tt:=nil;
 while cand_node <> nil do
  begin
   t:=cand_node; cand_node:=cand_node .next;
   if final_check(t,d,QQ,bb,cc) then
    begin t .ykx:=QQ; t .mx:=bb; t .xk:=cc; t .yk:=sqrt(d);
          t .next:=tt; tt:=t;  end;
  end;
 sort_solutions(tt);
 NS:=0;
 while tt <> nil do
  begin
   NS:=NS+1;
   t:=tt; tt:=tt .next;
   for i:=1 to t .p do cpd[NS,i]:=t .hx[i];
   for i:=t .p+1 to Nx do cpd[NS,i]:=0;
   ev[NS]:=t .ev;
   maxd[NS]:=t .yk;
   c[NS]:=t .xk;
   b[NS]:=t .mx;
   Q[NS]:=t .ykx;
  end;
end;
```

```
{--- Expand procedure ----------------------------}


procedure expand;
 label 10;
 var  Q          :arr3_3;
      b          :arr3;
      c,d,c1,c2  :real;
      t1,t2,t3   :ref_node;
      i,j,i1     :integer;
begin
 if open_node = nil then goto 999
  else begin t1:=open_node; open_node:=open_node .next; end;
 i1:=t1 .p+1;
 if i1 <= Nx then
  begin
   if (Nx-t1 .p+t1 .sz) > SIZE then
    begin
     t2:=new_node;
     t2 :=t1 ;
     t2 .p:=i1; t2 .hx[i1]:=0;
     t2 .next:=nil;
     insert_node(t2);
    end;
   for i:=Ny downto 1 do if (i in t1 .hy) then
    begin
     c_range_check(t1,i1,i,c1,c2);
     if c2 > 0 then
      begin
       t2:=new_node;
       t2 .sz:=t1 .sz+1;
       t2 .p:=i1;
       for j:=1 to t1 .p do t2 .hx[j]:=t1 .hx[j];
       t2 .hx[i1]:=i;
       t2 .hy:=t1 .hy-[i];
       t2 .cmin:=c1; t2 .cmax:=c2;
       if update_and_check(t1,t2) = false then dispose_node(t2)
        else if t2 .sz < SIZE then insert_node(t2)
        else if t2 .sz = SIZE then
         begin t3:=new_node; t3 :=t2 ;
               t3 .next:=cand_node; cand_node:=t3; insert_node(t2); end
        else begin
         if final_check(t2,d,Q,b,c) = false then goto 10;
```

```
          while cand_node <> nil do
           begin t3:=cand_node; cand_node:=cand_node .next;
                                                dispose_node(t3); end;
          cand_node:=new_node; cand_node :=t2 ;
          SIZE:=SIZE+1; writeln(SIZE);
          10:
          insert_node(t2);
         end;
       end;
     end;
   end;
 dispose_node(t1);
end;


{-- Main procedure ----------------------------------------}


begin
 cpu:=clock; h1:=0; h2:=0;
 SIZE:=3; Delta2:=Delta*Delta;
 compute_graph;
 init_node_list;
 while true do expand;
 999:
 writeln('Search Finished');
 found;
 cpu:=clock-cpu;
 writeln('  time =',cpu,'  update =',h1,'  final =',h2);
 writeln;
end;


end.
```

APPENDIX E. Shape Matching Program

```
MODULE  SM_2D(input,output);

type    arr2   = array[1..2] of real;

{
### Explanation of arguments ##############################################
 :: INPUT ::
    X  : List of model points along a contour.
    Y  : List of scene points along a contour.
    Nx : Number of model points.
    Ny : Number of scene points.
    MIRROR        : MIRROR=1 -> det(R)=1 or -1, MIRROR<>1 -> det(R)=1
    Delta         : Maximum distance of corresponding points.
    TH_a          : Threshold value for inner angles of corners (degrees).
    SW_c          : SW_c <> 1 ==> no constraint on c.
    min_c, max_c : Prespecified range of c. 0 <= min_c, max_c <= 1e10.
 :: OUTPUT ::
    NS       : Number of obtained matchings.
    cpd      : Matching result (list of correspondences)
                  cpd[i,X_index] = Y_index.
    ev       : The least mean squared error.
    maxd     : Maximum distance.
    c        : Scaling.
    b        : Translation.
    Q        : Rotation (degrees).
                  Rotation matrix = |  cos(Q)       sin(Q)     |
                                    | -detQ*sin(Q)  detQ*cos(Q) |
    detQ     : det(Q).
    cpu      : cputime.
 }


 [GLOBAL] procedure sm_2D
             (var X    :array[x1..x2 :integer] of arr2;
              var Y    :array[y1..y2 :integer] of arr2;
                  Nx   :integer;
                  Ny   :integer;
                  MIRROR   :integer;
                  Delta    :real;
                  TH_a     :real;
```

```
            SW_c      :integer;
            min_c     :real;
            max_c     :real;
        var NS    :integer;
        var cpd   :array[cpd1..cpd2 :integer; cpd3..cpd4 :integer]
                                                    of integer;
        var ev    :array[ev1..ev2 :integer] of real;
        var maxd  :array[maxd1..maxd2 :integer] of real;
        var c     :array[c1..c2 :integer] of real;
        var b     :array[b1..b2 :integer] of arr2;
        var Q     :array[q1..q2 :integer] of real;
        var detQ  :array[detq1..detq2 :integer] of integer;
        var cpu   :integer);


label 999;

const  maxNy    = 100;
       maxNx    = 50;

type   arr2_2     = array[1..2,1..2] of real;
       arrN       = array[1..maxNy] of real;
       byte       = 0..255;
       ref_node   =  node;
       node       = record
                      sz      :integer;
                      p       :integer;
                      f,c     :integer;
                      hx      :packed array[1..maxNx] of byte;
                      cmin,cmax :real;
                      mx,my   :arr2;
                      xk,yk   :real;
                      ykx     :arr2_2;
                      ev      :real;
                      next    :ref_node;
                    end;

var    open_node  : ref_node;
       cand_node  : ref_node;
       garbage    : ref_node;
       SIZE       : integer;
       Delta2     : real;

       Gx         : array[1..maxNx,1..maxNx] of real;
       Gy         : array[1..maxNy,1..maxNy] of real;
```

```
        Ax,Ay       : arrN;

        h1,h2,h3    : integer;

{-- SVD ----------------------------------------}

{ Fortran subroutine in LINPACK.
  This subroutine must be linked when linking programs.
  Ex. $LINK main_program,2dsm,linkpack/lib            }

[EXTERNAL] procedure ssvdc
               (%ref X    :array[x1..x2 :integer; x3..x4 :integer] of real;
                %ref LDX  :integer;
                %ref N    :integer;
                %ref P    :integer;
                %ref S    :array[s1..s2 :integer] of real;
                %ref E    :array[e1..e2 :integer] of real;
                %ref U    :array[u1..u2 :integer; u3..u4 :integer] of real;
                %ref LDU  :integer;
                %ref V    :array[v1..v2 :integer; v3..v4 :integer] of real;
                %ref LDV  :integer;
                %ref WORK :array[w1..w2 :integer] of real;
                %ref JOB  :integer;
                %ref INFO :integer);FORTRAN;


procedure svd_1(var A :arr2_2; var trd :real);
begin
 if MIRROR = 1 then
    trd:=sqrt(A[1,1]**2+A[1,2]**2+A[2,1]**2+A[2,2]**2
              +2*abs(A[1,1]*A[2,2]-A[1,2]*A[2,1]))
  else
    trd:=sqrt((A[1,1]+A[2,2])**2+(A[1,2]-A[2,1])**2)
end;


procedure svd_2(A :arr2_2; var Q :arr2_2; var  trd :real);
 var  U,V         :arr2_2;
      D,E,WORK    :arr2;
      info,i,j,k :integer;
      r,r1,r2     :real;
begin
 r:=A[1,1]*A[2,2]-A[1,2]*A[2,1];
 ssvdc(A,2,2,2,D,E,U,2,V,2,WORK,11,info);
```

```
 if info <> 0 then writeln(' ERROR -- SVD ');
 if MIRROR = 1 then trd:=D[1]+D[2]
 else if r >= 0 then trd:=D[1]+D[2]
  else begin trd:=D[1]-D[2]; U[2,1]:=-U[2,1]; U[2,2]:=-U[2,2]; end;
 for i:=1 to 2 do for j:=1 to 2 do
   begin Q[i,j]:=0; for k:=1 to 2 do Q[i,j]:=Q[i,j]+V[k,i]*U[k,j]; end;
end;


{--- Miscellaneous ------------------------------}

function angle_Q(var Q :arr2_2):real;
 var  th,c,s  :real;
begin
 c:=Q[1,1]; s:=Q[1,2];
 if abs(c) < 1e-4 then
  begin if s >= 0 then th:=90 else th:=-90 end
  else begin
   th:=arctan(s/c)*180/3.1416;
   if c < 0 then
    if s >= 0 then th:=th+180 else th:=th-180;
  end;
 angle_Q:=th;
end;


procedure apply_Qbc(Q :arr2_2; b :arr2; c :real; x1 :arr2; var x2 :arr2);
 var  i,j  :integer;
begin
 for i:=1 to 2 do
  begin x2[i]:=b[i];
        for j:=1 to 2 do x2[i]:=x2[i]+c*Q[i,j]*x1[j]; end;
end;


procedure compute_graph;
 var  i,j  :integer;
begin
 for i:=1 to Nx-1 do for j:=i+1 to Nx do
  begin Gx[i,j]:=sqrt((X[i,1]-X[j,1])**2+(X[i,2]-X[j,2])**2);
        Gx[j,i]:=Gx[i,j]; end;
 for i:=1 to Ny-1 do for j:=i+1 to Ny do
  begin Gy[i,j]:=sqrt((Y[i,1]-Y[j,1])**2+(Y[i,2]-Y[j,2])**2);
        Gy[j,i]:=Gy[i,j]; end;
end;
```

```
procedure product_range(var c1,c2,c3,c4 :real);
 { [c1,c2] * [c3,c4] ==> [c1,c2] , if c2 < 0 then []*[] = null }
begin
 if c1 <= c3 then
  begin
   if c2 < c3 then c2:=-1
    else if c4 <= c2 then begin c1:=c3; c2:=c4; end
                     else c1:=c3;
  end else begin
   if c4 < c1 then c2:=-1
    else if c2 > c4 then c2:=c4;
  end;
 end;


 {--- Node list handling  ----------------------}

 function  new_node:ref_node;
  var t  :ref_node;
 begin
  if garbage = nil then begin new(t); t .next:=nil; new_node:=t; end
   else begin t:=garbage; garbage:=garbage .next;
              t .next:=nil; new_node:=t;        end;
 end;


 procedure dispose_node(t :ref_node);
 begin
  t .next:=garbage; garbage:=t;
 end;


 procedure insert_node(var t :ref_node);
 begin
  t .next:=open_node;
  open_node:=t;
 end;


 procedure init_node_list;
  var  t   :ref_node;
       i,j :integer;
```

```
begin
 cand_node:=nil; garbage:=nil; open_node:=nil;
 new(t);
 with t  do
  begin
   sz:=0; p:=0;
   f:=0; c:=0;
   if SW_c = 1 then begin cmin:=min_c; cmax:=max_c; end
    else begin cmin:=0; cmax:=1e10; end;
   mx[1]:=0; mx[2]:=0; my[1]:=0; my[2]:=0; xk:=0; yk:=0;
   for i:=1 to 2 do for j:=1 to 2 do ykx[i,j]:=0;
   ev:=0; next:=nil;
  end;
 open_node:=t;
end;


{--- Angle calculation ------------------------------}


procedure calc_angle(var D :array[d1..d2 :integer; d3..d4 :integer]
                                                              of real;
                     N :integer; var ad :arrN);
 var  i,ip,is  :integer;
      v1,v2    :arr2;

     function angle_bv(v1,v2 :arr2):real;
      var  s,c,th  :real;
     begin
      s:=v1[1]*v2[2]-v1[2]*v2[1];
      c:=v1[1]*v2[1]+v1[2]*v2[2];
      if abs(c) < 1e-4 then
       begin if s >= 0 then th:=90 else th:=-90 end
       else begin
        th:=arctan(s/c)*180/3.1416;
        if c < 0 then
         if s >= 0 then th:=th+180 else th:=th-180;
       end;
      angle_bv:=th+180;
     end;

 begin
  for i:=1 to N do
   begin
    if i = 1 then ip:=N else ip:=i-1;
```

```
   if i = N then is:=1 else is:=i+1;
   v1[1]:=D[i,1]-D[ip,1]; v1[2]:=D[i,2]-D[ip,2];
   v2[1]:=D[is,1]-D[i,1]; v2[2]:=D[is,2]-D[i,2];
   ad[i]:=angle_bv(v1,v2);
  end;
end;

{--- Update and check ---------------------------}

function update_and_check(t1,t2 :ref_node):boolean;
 label 10;
 var  i,j,k,k1,k2,k3  :integer;
      c,d,r           :real;
      xx,yy,b         :arr2;
      Q               :arr2_2;
      f               :boolean;
begin
 h1:=h1+1;
 k1:=t2 .sz-1; k2:=t2 .p; k3:=t2 .hx[k2];
 for i:=1 to 2 do
  begin
   t2 .mx[i]:=(t1 .mx[i]*k1+X[k2,i])/(k1+1);
   t2 .my[i]:=(t1 .my[i]*k1+Y[k3,i])/(k1+1);
   xx[i]:=t1 .mx[i]-X[k2,i];
   yy[i]:=t1 .my[i]-Y[k3,i];
  end;
 r:=xx[1]**2+xx[2]**2;
 t2 .xk:=t1 .xk+r*k1/(k1+1);
 r:=yy[1]**2+yy[2]**2;
 t2 .yk:=t1 .yk+r*k1/(k1+1);
 for i:=1 to 2 do for j:=1 to 2 do
   t2 .ykx[i,j]:=t1 .ykx[i,j]+yy[i]*xx[j]*k1/(k1+1);
 svd_1(t2 .ykx,d);
 t2 .ev:=(t2 .yk-(d**2)/t2 .xk)/(k1+1);
 f:=true;
 if t2 .ev > Delta2 then begin f:=false; goto 10; end;
 if SW_c = 1 then
   begin c:=d/t2 .xk;
         if (c < t2 .cmin) or (c > t2 .cmax) then f:=false; end;
 10:
 update_and_check:=f;
end;
```

```
function final_check(t :ref_node; var maxd :real;
                     var Q :arr2_2; var b :arr2; var c :real):boolean;
 label 10;
 var   xx    :arr2;
       d,r   :real;
       i,j,k   :integer;
begin
 h2:=h2+1;
 maxd:=0;
 svd_2(t .ykx,Q,d);
 c:=d/t .xk;
 for i:=1 to 2 do
  begin b[i]:=t .my[i];
        for j:=1 to 2 do b[i]:=b[i]-c*Q[i,j]*t .mx[j]; end;
 for i:=1 to t .p do
  begin
   k:=t .hx[i];
   if k <> 0 then
    begin
     apply_Qbc(Q,b,c,X[i],xx);
     r:=(xx[1]-Y[k,1])**2+(xx[2]-Y[k,2])**2;
     if r > Delta2 then begin final_check:=false; goto 10 end;
     if r > maxd then maxd:=r;
    end;
  end;
 final_check:=true;
 10:
end;


procedure c_range_check(t :ref_node; ix,iy :integer; var c1,c2 :real);
 label 10;
 var  c3,c4  :real;
      i      :integer;

begin
 c1:=t .cmin; c2:=t .cmax;
 if t .sz <> 0 then
  for i:=1 to t .p do if t .hx[i] <> 0 then
   begin
    c3:=(Gy[t .hx[i],iy]-2*Delta)/Gx[i,ix];
    c4:=(Gy[t .hx[i],iy]+2*Delta)/Gx[i,ix];
    if c3 < 0 then c3:=0;
    product_range(c1,c2,c3,c4);
```

```
    if c2 < 0 then goto 10;
   end;
 10:
end;


{--- Found ------------------------------}

procedure found;
 var  QQ      :arr2_2;
      i       :integer;
      d,cc    :real;
      bb      :arr2;
      t,tt    :ref_node;

     procedure insert(tt :ref_node; var t :ref_node);
     begin
      if t = nil then t:=tt
       else if tt .yk < t .yk then begin tt .next:=t; t:=tt; end
       else insert(tt,t .next);
     end;

     procedure sort_solutions(var t :ref_node);
      var  tt  :ref_node;
     begin
      if t <> nil then
       begin
        tt:=t; t:=t .next; tt .next:=nil;
        sort_solutions(t);
        insert(tt,t);
       end;
     end;

 begin
  tt:=nil;
  while cand_node <> nil do
   begin
    t:=cand_node; cand_node:=cand_node .next;
    if final_check(t,d,QQ,bb,cc) then
     begin t .ykx:=QQ; t .mx:=bb; t .xk:=cc; t .yk:=sqrt(d);
           t .next:=tt; tt:=t; end;
   end;
  sort_solutions(tt);
  NS:=0;
  while tt <> nil do
```

```
  begin
   NS:=NS+1;
   t:=tt; tt:=tt .next;
   for i:=1 to t .p do cpd[NS,i]:=t .hx[i];
   for i:=t .p+1 to Nx do cpd[Ns,i]:=0;
   ev[NS]:=t .ev;
   maxd[NS]:=t .yk;
   c[NS]:=t .xk;
   b[NS]:=t .mx;
   with t  do d:=ykx[1,1]*ykx[2,2]-ykx[1,2]*ykx[2,1];
   Q[NS]:=angle_Q(t .ykx);
   if d > 0 then detQ[NS]:=1 else detQ[NS]:=-1;
  end;
end;


{--- Node expansion ----------------------------}


procedure expand;
 label 10;
 var  t1,t2,t3     :ref_node;
      Q            :arr2_2;
      b            :arr2;
      c,d,c1,c2      :real;
      i,j,i1,cf    :integer;

     procedure create_node1(var t1,t2 :ref_node; k :integer);
      var  i,i1 :integer;
     begin
      t2:=new_node;
      i1:=t1 .p+1;
      t2 .sz:=1; t2 .p:=i1;
      for i:=1 to t1 .p do t2 .hx[i]:=0;
      t2 .hx[i1]:=k; t2 .f:=k; t2 .c:=k;
      t2 .cmin:=t1 .cmin; t2 .cmax:=t1 .cmax;
      t2 .mx:=X[i1]; t2 .my:=Y[k];
      t2 .xk:=0; t2 .yk:=0; t2 .ykx:=t1 .ykx; t2 .ev:=0;
     end;

     procedure create_node2(var t1,t2 :ref_node; k :integer);
      var  i  :integer;
     begin
      t2:=new_node;
      t2 .sz:=t1 .sz+1;
```

```
      t2 .p:=t1 .p+1;
      for i:=1 to t1 .p do t2 .hx[i]:=t1 .hx[i];
      t2 .hx[t2 .p]:=k;
      t2 .f:=t1 .f;
      t2 .c:=k;
      t2 .cmin:=c1; t2 .cmax:=c2;
     end;

begin
 if open_node = nil then goto 999
  else begin t1:=open_node; open_node:=open_node .next; end;
 i1:=t1 .p+1;
 if i1 <= Nx then
  begin
   if (Nx-t1 .p+t1 .sz) > SIZE then
    begin
     t2:=new_node;
     t2 :=t1 ;
     t2 .p:=i1; t2 .hx[i1]:=0;
     t2 .next:=nil;
     insert_node(t2);
    end;
   if t1 .sz = 0 then
     begin
      for i:=1 to Ny do
       if abs(Ax[i1]-Ay[i]) < TH_a then
        begin create_node1(t1,t2,i); insert_node(t2); end;
     end else begin
      i:=SIZE-t1 .sz-1;
      if t1 .f > i then cf:=t1 .f-i else cf:=t1 .f-i+Ny;
      i:=t1 .c+1; if i > Ny then i:=1;
      repeat
        if abs(Ax[i1]-Ay[i]) < TH_a then
         begin
          h3:=h3+1;
          c_range_check(t1,i1,i,c1,c2);
          if c2 > 0 then
           begin
            create_node2(t1,t2,i);
            if update_and_check(t1,t2) = false then dispose_node(t2)
             else if t2 .sz < SIZE then insert_node(t2)
             else if t2 .sz = SIZE then
              begin t3:=new_node; t3 :=t2 ;
                    t3 .next:=cand_node; cand_node:=t3; insert_node(t2); end
             else begin
```

```
            if final_check(t2,d,Q,b,c) = false then goto 10;
            while cand_node <> nil do
             begin t3:=cand_node; cand_node:=cand_node .next;
                   dispose_node(t3); end;
            cand_node:=new_node; cand_node :=t2 ;
            SIZE:=SIZE+1; writeln(SIZE);
            10:
            insert_node(t2);
           end;
         end;
       end;
      i:=i+1; if i > Ny then i:=1;
     until i = cf;
   end;
  end;
 dispose_node(t1);
end;


{--- Main procedure ----------------------------------------}


begin
 cpu:=clock; h1:=0; h2:=0; h3:=0;
 SIZE:=3; Delta2:=Delta*Delta;
 calc_angle(X,Nx,Ax); calc_angle(Y,Ny,Ay);
 compute_graph;
 init_node_list;
 while true do expand;
 999:
 writeln('Finished');
 found;
 cpu:=clock-cpu;
 writeln('   time =',cpu,'   check =',h3,'  update =',h1,'  final =',h2);
 writeln;
end;


 end.
```